GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[X] ORIGINAL [ ] REVISION NO. _____

Project No. E-25-M21 (R6227-0A0) GTRC/~~GIT~~ XXX DATE 10 / 29 / 86

Project Director: R. E. Fulton School/~~Lab~~ XXXX ME

Sponsor: MacNeal-Schwendler Corporation

Type Agreement: Research Project Agreement dated 10/10/86

Award Period: From 9/22/86 To ~~9/21/87~~ 9/21/88 (Performance) 10/15/87 (Reports)

Sponsor Amount:	This Change	Total to Date
Estimated: $		$ 65,603
Funded: $		$ 65,603

Cost Sharing Amount: $ _____ Cost Sharing No: N/A

Title: Structural Dynamics Methods for Parallel Supercomputers

ADMINISTRATIVE DATA — OCA Contact John B. Schonk X4820

1) Sponsor Technical Contact:

2) Sponsor Admin/Contractual Matters:

J. F. Gloudeman
MacNeal-Schwendler Corp.
815 Colorado Blvd.
Los Angeles, CA 90041-1777

Defense Priority Rating: N/A Military Security Classification: N/A

(or) Company/Industrial Proprietary: N/A

RESTRICTIONS

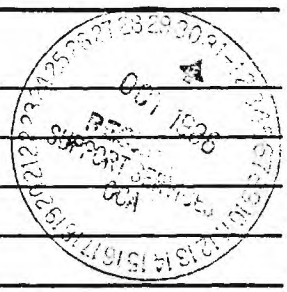
See Attached N/A Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with Sponsor, however none proposed

COMMENTS:

The Sponsor has agreed to pay an advance payment of $16,400.



COPIES TO:

SPONSOR'S I. D. NO. _____

Project Director	Procurement/GTRI Supply Services	GTRC
Research Administrative Network	Research Security Services	Library
Research Property Management	Reports Coordinator (OCA)	Project File
Accounting	Research Communications (2)	Other A. Jones

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Date 3/15/89

Project No. E-25-M21 Center No. R6227-0A0

Project Director R. E. Fulton School/Lab ME

Sponsor MacNeal-Schwendler Corporation

Contract/Grant No. Agreement dtd 10/10/86 GTRC XX GIT ____

Prime Contract No. _____

Title Structural Dynamics Methods for Parallel Supercomputers

Effective Completion Date 2/15/89 (Performance) 2/15/89 (Reports)

Closeout Actions Required:

____ None
____ Final Invoice or Copy of Last Invoice - Already Submitted
X Final Report of Inventions and/or Subcontracts - Questionnaire sent to P.I.
____ Government Property Inventory & Related Certificate
____ Classified Material Certificate
____ Release and Assignment
____ Other _____

Includes Subproject No(s). _____

Subproject Under Main Project No. _____

Continues Project No. _____ Continued by Project No. _____

Distribution:

X Project Director
X Administrative Network
X Accounting
X Procurement/GTRI Supply Services
X Research Property Management
___ Research Security Services
X Reports Coordinator (OCA)
X GTRC
X Project File
2 Contract Support Division (OCA)
___ Other _____

E-25-M21

FINAL REPORT
Phase 1

STRUCTURAL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

By:
Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

Submitted to:
The Mac-Neal Schwendler Corporation
Los Angeles, California

Under:
Georgia Tech Research Corporation E25-M21

October, 1987

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF MECHANICAL ENGINEERING
ATLANTA, GEORGIA 30332

1987

STRUCTURAL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

By

Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

Final Report Phase I
Submitted to

The Mac-Neal Schwendler Corporation
Los Angeles, California

Georgia Tech Research Corporation E25-M21
to Mac-Neal Schwendler Corporation

October, 1987

STRUCTURAL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

by
Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

TABLE OF CONTENTS

PARALLEL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

SECTION I

INTRODUCTION

The structural design/analysis of future advanced systems in such fields as transportation, energy and aerospace require increasing levels of detail and design complexities to meet mission requirements. Such required analysis may include the combined effects of dynamic loads, composite materials, multidisciplinary interactions, three dimensional geometry and nonlinear behavior. Realistic structural models for such designs typically imply large-order finite element models and excessively large computational requirements. Since the advent of NASTRAN (1), general-purpose finite element structural analysis computer programs (e.g., 2) have provided the capability to address a wide range of structures problems. However, when these programs are applied to nonlinear problems, the limitations of sequential computers result in reduced scale models or excessive computing times. For example, the routine dynamic analysis of large scale structural problems is not feasible on current sequential computers, and such computers are inadequate to support analysis requirements for many future engineering designs where effective speeds greater than 10^3 MFLOPS (million floating point operations per second) will be required. For example, calculations for such problems as nonlinear dynamic response, multidisciplinary interactions and optimum design are all severely limited by computation speed, and models are usually restricted to small-order problems or highly simplifying approximations.

Projected advances in computer technology indicate significant increases in effective calculation speed will be available in the 1990's, through fifth generation super computer architectures consisting of arrays of processors operating in parallel on different tasks (see e.g. Ref. 3 for a survey). Such advanced supercomputers denoted MIMD (Multiple Instruction, Multiple Data) computers have the potential for increasing effective calculation speeds by several orders of magnitude (Figure 1). But this potential increase in speed cannot be effectively utilized without the development and implementation of appropriate numerical algorithms for structures which take advantage of the parallel computation features of this new generation of computers. Use of existing conventional algorithms and software will not realize the full potential of these new MIMD computers, and research is needed on the development of parallel structural analysis/design algorithms for these computers. One critically needed research effort is the development and evaluation of parallel methods for dynamic analyses of complex nonlinear finite element structural problems.

The research summarized herein reports on an investigation and evaluation of selected algorithms for MSC/NASTRAN which appear best suited for nonlinear dynamic analyses on parallel supercomputers. The principal areas of investigation are

1. An investigation of the architectural characteristics of future parallel computers relative to finite element methods.

2. Software which implements specific nonlinear finite element methods and is tested on a commercial parallel computer. The implementation utilizes MSC/NASTRAN computational and parallel kernels, and emulates the MSC/NASTRAN I/O environment.
3. Validation of the most promising methods through application to test problems and measurement of computing speedups due to parallelism.
4. A study of candidate methods for nonlinear dynamic analysis on a parallel computer to provide a basis for future software implementation.

The following sections address each of these areas. Section II covers the first area, Section III addresses the second and third areas and Section IV addresses the fourth. Specific summaries or conclusions relative to each of these areas is included at the end of the respective sections.

SECTION II
GROWTH OF PARALLEL COMPUTERS

2.1 INTRODUCTION

New developments in computer technology are making obsolete the traditional computational environments employed by the scientific/engineering community. The information-oriented society of tomorrow will increasingly rely upon computers as tools for supporting the creative process in industrial and social planning. The computational power demanded by todays Computer Aided Engineering and Design tools are large enough to cause the Von-Neumann type of computers with sequential processing to reach practical limits in speed. Technological advances in components design and manufacturing alone cannot meet the computational requirements of the scientific community. It is in this context that the need for special purpose architecture and hardware/software methodology becomes highlighted. Research and development efforts in this area are well underway in the industrialized nations; for example the DARPA Strategic Computing and Survivability Project in the US, and the Supercomputer and Fifth Generation computer projects in Japan. The research efforts conducted in the private sector have already led to the development of pipeline and vector processing architectures that employ multiple arithmetic pipeline units and powerful vectorizing compilers. However, it is being recognized that the speed of pipeline architectures is nearing the practical limit and that true parallel processing/multicomputing is essential to achieve ultra-high speed computing. It is imperative that parallel processing architectures be complemented by a fully user

programmable software development environment to improve productivity. The Multiple Instruction Multiple Data (MIMD) architecture, a true multicomputing architecture, is an example of such an environment and has been extremely popular with the scientific computering community. Figure 2 shows the various interconnection topologies employed in Multicomputing and Figure 3 shows two information sharing mechanisms used. A particularly attractive topology for industrial and business applications is the bus architecture with a shared memory such as the FLEX/32 multicomputer discussed later. A topology well suited for high speed image processing is the PIXAR vector and parallel computer discussed later. This report provides a brief discussion of the parallel processing issues, and an overview of several parallel computing systems.

2.2 PARALLEL PROCESSING

Parallel processing is the technique of achieving "true concurrency" wherein two or more sequential processes are executed on separate computers at exactly the same instant of time. Parallel processing is also denoted multicomputing, but is different from multitasking which involves the interleaved or simultaneous execution of two or more processes by a single computer. An ideal environment for parallel programming would be a collection of tightly coupled yet autonomous computers capable of synchronizing and communicating in parallel, but also capable of operating independently. Each processor should have its own memory and I/O so that true concurrency can be achieved. The user should be able to employ software constructs which achieve synchronization between processors as well as

operate on shared data in a mutually exclusive fashion in order to maintain data integrity. It should be noted that an ideal parallel architecture for engineering/scientific computations is the one that is fully user-programmable via software constructs, as opposed to those that employ compilers instead of software constructs. Another essential feature for true multicomputing is the existence of shared memory where complex data structures such as those found in a finite element database could be stored.

Programming in a multicomputing environment introduces added complexity in the software design and algorithmic strategy. The total hardware/software system must be taken in to consideration when developing and implementing a parallel algorithm. The implementation criteria that influence the efficiency of an algorithm include the amount of communication versus the amount of computation in a given problem, the balance of workload among the processors, the communication paths and synchronization delays, and the size of the problem relative to the number of processors used. The flow of the algorithm should be analyzed to identify those calculations which must be sequential and those which can be done in parallel. Efficient algorithms typically maximize parallel calculations, minimize sequential calculations, minimize communication, and partition tasks onto a processor array so that communication paths are effective. Implementing an existing system brings further complexities in that an evolutional modularized development strategy is essential to achieve progress in a step by step fashion, to partition work assignments and to

maintain a stable baseline software. A modular development strategy is important in a sequential computer but it can be even more important in a parallel environment. For example a parallel computer implementation can make code debugging more difficult than that for a sequential computer especially when parallel tasks are carried out asynchronously. To improve reliability in parallel processing the issues to be dealt with include

- Multiple Processors and Memories
- Repeatability of results
- Trail of calculations
- Configuration management of data
- Non-standard parallel languages

These issues can be readily dealt with through software development strategies which consist of

- Software modularity
- Centralized data base management
- Code sequentially first & use good coding practices

2.3 COMMERCIAL PARALLEL PROCESSORS

Some thirty vendors have introduced "parallel machines" during the last three years. While some of these machines may not perform "true" parallel processing, the number of parallel processor machines available in the market is rapidly increasing. Cray Research already markets the CRAY-2 and CRAY-XMP with four vector processors; the CRAY-YMP with 8 processors is soon to be released and a CRAY-3 with 32 processor is projected

by 1990. At the other extreme Alliant offers the Alliant FX8 super mini-computer with 8 processors and AT&T recently announced its PXM 900 series high performance image systems for 3D graphics and image processing. In the following sections, some of the MIMD machines (both shared memory and distributed multiprocessors) introduced to the market recently are reviewed; a more detailed review of parallel processing is given in (4), (5), (6) and (7).

2.3.1 Sequent S Series

Sequent Computer Systems recently introduced two UNIX-based S (Symmetry) series machines (S27 and S81) in their line of Practical Parallel Computers with a price range of $90,000 to $800,000 (8). These S series machines are single-bus shared memory systems that use the Intel 16 MHz 80386 CPUs. Each CPU includes a 80387 floating point co-processor, a 64 KB two-way set-associative cache, and a 64-bit bus interface logic operating at 80 MB per second. For a long time the consensus has been that shared-memory multiprocessor computers are impractical because the addition of each processor improves the overall performance marginally and not linearly due to the increased competition for the shared resources such as the memory and the system bus.

To overcome this difficulty, Sequent claims achieving close to linear performance through the special design of the local processor cache, memory, system bus, synchronization mechanisms, and other components. Sequent's S machines are not based on the master/slave organization. They are based on a tightly-coupled symmetrical system that

distribute the processing load dynamically to an arrangement of two to thirty CPUs. All CPUs can execute application programs as the users perform I/O or other service functions. The multiprocessing operating system is Dynix 3.0 (Dynamic load-balancing UNIX) that can accommodate FORTRAN, Pascal, C and Ada.

The S27 configuration includes two to ten 80386 CPUs, 8 to 80 MB of main memory, 150 MB to 4.3 GB of disk storage, and up to 96 RS-232 serial connections with a prive range of $90,000-$450,000. The S81 configuration includes two to thirty 80386 CPUs, 8 to 240 MB of main memory, 264 MB to 17.2 GB of disk storage, and up to 256 serial connections with a price range of $164,000-$800,000. The S machines are not intended to compete with vector-intensive applications and Cray-like machines.

2.3.2 Encore Multimax

In 1985, Encore corporation introduced its Multimax shared memory multiprocessor (9). It is a single-bus system and uses up to 20 National Semiconductor 32032 32-bit processors with LSI memory management units and floating point co-processors. Each processor is capable of executing 0.75 MIPS. Thus the total processing power of a high-end Multimax is 15 MIPS. It can accommodate up to 32 Mbytes of shared memory. The system is configured in terms of dual processor cards each containing two processors and a 32 Kbyte cache shared between the two processors. Multimax uses a fast bus, called Nanobus, which can carry 96 bits of new information every 80 nanosec, even if previous requests are still in progress. The result is a true data transfer bandwidth of 100 Mbytes/second. Encore is in the same price range as Sequent.

Encore Multimax supports two operating systems Umax 4.2 and Umax V which are based on Unix 4.2 and Unix V, respectively. Programs running on different processors can get simultaneous access to operating system services from a single shared copy of UMAX. By using a technique called multi-threading which implements multiple simultaneous streams of control, UMAX can process many requests in parallel.

2.3.3 FLEX/32

To avoid traffic congestion, bus-based machines are usually a limited to a moderate number of processors (usually less than 20). To overcome this difficulty, Flexible Computer Corporation developed the Flex/32 on a design concept referred to as a multicomputer (10,11). In this architecture, in addition to a global shared memory, each 32 bit processing node has its own local memory and I/O facility, and runs its own operating systems (Figure 4).

Each of the processors are directly programmable and the FLEX/32 has an open architecture in that it uses non-proprietary I/O devices. It has unlimited expansion in memory and power and is easily re-configured. FLEX/32 supports the UNIX system V, as a software development environment to create, compile, simulate and execute parallel algorithms. In addition Flexible/32 supports a real time operating system, MMOS, for true concurrent processing. Other software tools include a concurrency simulator, high level programming languages such as FORTRAN, C and ADA, and effective software constructs for interprocessor communication. The concurrency simulator runs under UNIX system V and is used for debugging the process of interprocessor synchronization. The concurrency

simulator is also an excellent tool to be used as a mid-step in the phased migration of concurrent code present. At present, the largest FLEX/32 in the market (at MCC) uses 40 processors; Flexible Computer Corporation claims that the machine can theoretically use up to 20000 processors. Another feature of Flex/32 is that different types of processors can be mixed in the machine. At present, two processor modules are available, one based on the Motorola 68020 and the other based on Semiconductor 32032 microprocessors; the company has demonstrated both microprocessors operating simultaneously in one computer, the only known heterogeneous parallel computer.

2.3.4. Intel Parallel Supercomputer

Intel Parallel SuperComputer or iPSC consists of 16, 32, 64, or 128 high-performance microcomputers (12). Each microcomputer (currently an Intel 80286 CPU with an upgrade to the 80386 underway) and its associated local memory (512 Kbytes of NMOS Dynamic RAM) together with a numeric processing unit (intel 80287) is called a node. The interconnection network is a hypercube.

The nodes are connected with each other through high-speed bi-directional communication channels forming a self-contained "cube" in a free-standing enclosure. A "cube manager" connected to the cube includes the XENIX operating system which supports program development tools (e.g., C and FORTRAN compilers), and commands to manage the cube and to monitor its status. There is no global shared memory. Each node has its local memory and therefore is completely independent. Nodes communicate with their neighbors through queued message passing. A built-in timer measures the processing

(computation plus communication) time of each processor.

iPSC is used primarily for numerical analysis and computer science research associated with the development of parallel processing algorithms. It can be used as a stand-alone concurrent machine or as a server in a distributed processing environment through an Ethernet interface. Intel has recently introduced iPSC-VX in which a vector processor is attached to each CPU. This machine is claimed to have a peak performance of 424 MFLPOPS. In contrast, the peak performance of the four processor supercomputer Cray X-MP48 is around 1000 MFLOPS.

2.3.5. Transputers

Most parallel machines except the FLEX/32 typically have fixed architectures while simple alterations such as replacing a malfunctioning board can be done by the purchaser. These machines cannot be readily reconfigured in a way suited best for a given application. Basically they are meant to be used by mapping the application onto the fixed architecture. In some cases such as the Alliant the user has little or no control on how the mapping takes place.

INMOS' Transputers (13, 14) have been designed to facilitate reconfiguration of parallel machines to meet the interconnection requirements of a given application. A transputer is a single chip containing a 32-bit RISC-like microprocessor, 2K bytes of on-chip memory, memory interface to access up to 4G bytes of off-chip memory, and 4 bidirectional communication links which provide point-to-point communication between the transputers. The four communication links coming out of a transputer look like telephone jacks and

different transputers can be connected simply by plugging-in the links together.

Though transputers can be connected in arbitrary ways, one type of interconnection is naturally suggested by the transputer architecture and has certain advantage over the hypercube. INMOS currently supplies a board with 4 transputers and 256 Kbytes of memory local to each transputer with a cost of about $5,000. On this board the four transputers are hardwire-connected in a square. If the diagonals are also connected (manually), the resulting board looks like a super transputer because it has four communication links which can be used to connect it to other transputer boards. Four such boards may be connected in a similar fashion to yield a 16-node super transputer. If one uses the term T-net (T for transputer) for this type of interconnection then a single transputer is a zero dimensional T-net, the 4-transputer board is a 1-dimensional T-net, and the 16-node transputer system is a 2-dimensional T-net. It should be noted that an n-dimensional T-net has 4^n nodes and its diameter d_n is given by $d_n = 2*d_{n-1} + 1$. The diameter of this network grows faster than that of a hypercube but the advantage of T-net is that it has a constant connectivity (which is currently 4) irrespective of its dimension.

2.3.6 PIXAR

An especially interesting parallel computer for image processing is the PIXAR Image computer developed by PIXAR (Figure 5). This system provides a computer architecture tailored to performing image computing operations (15). It uses a closely-coupled host computer to provide those functions not directly related to image computing; typical hosts

include (Figure 4)

- Sun 3/160, Sun 3/180, Sun 3/260, Sun 3/280
- Silicon Graphics IRIS 3100 Series
- DEC Micro VAXII/Ultrix

PIXAR software is developed under the UNIX 4.2 operating system in both C and assembly language. The PIXAR Image Computer consists of a 21 inch-high, rack-mounted box with 12 slots for printed-circuits boards. The minimum system contains six boards: one Chap, one video, one memory controller, and three 8 Mbyte memory boards. It is expandable to three Chaps, two video and six memory boards. The host can be up to 30 feet from the PIXAR Image Computer. The host connection is made through an interface card that plugs into the host's I/O bus. Figure 5b shows the various boards and communication paths in a PIXAR environment. The interface card connects the host bus, which might be VME or Multibus, to the SYSbus; a 16-bit 2Mbyte/second bandwidth bus over which instructions and data are sent to the Pixar Image Computer.

As shown in Figure 5b the YAPbus is an 80 Mbytes/sec high-bandwidth bus connection to a wide range of peripherals. The Pixar Image Computer primarily consists of a fast processor tightly coupled to a large memory. As shown in Figure 5b the Chap communicates with the memory over the Pbus. This 240 Mbytes/second bandwidth bus provides the tight coupling between the memory and the processor. The Chap processor receives instructions from the host computer, controls and YaAPbus, and computes on the data in the image memory.

The video board reads image data out of the memory to refresh the display. Data transfers from the memory to the video board over the Vbus, which has a bandwidth of 480 Mbyte/second bandwidth of the memory system. The memory controller receives requests for memory resources from the Chaps or video board. It then arbitrates and schedules the data transfers when the appropriate resources are available.

All the components of a Pixar Image Computer are designed to accommodate the pixel data structure. Full-color pixels are stored in memory as four 12-bit quantities, one each for the Red, Green, Blue, and Alpha components, or "channels", of a picture. Together, these four channels define the color (RGB) and transparency (Alpha) of any particular pixel in memory. Monochrome applications treat the Alpha channel as a fourth framebuffer; other applications might use the Alpha channel as a scratch space. Images are stored on the disk as four separate pictures allowing the programmer to use the memory in many different ways.

The Chap is a SIMD machine - for Single Instruction, Multiple Data. The Chap has one processor for each of the Red, Green, Blue, and Alpha channels. Four processors execute the same instruction on four values simultaneously. These four values can be the four components of a single pixel (RGBA), four entries in a table, the same component from four adjacent pixels, etc. Since each processor runs at 10 MIPS, the total speed is 40 MIPS.

The Chap contains two types of hardware elements: vector and scalar. The vector elements are sonamed because there are four of each element: one for the Red, Green,

Blue and Alpha channels. There is only one of each scalar element per Chap.

2.4 PARALLEL COMPUTER REQUIREMENTS

Figure 6 lists some of the requirements of a parallel computer to support computer science related research. Figure 7 lists requirements to support finite element and other engineering applications. Figure 8 illustrates the philosophy of phased migration of code involving three major steps. These steps are to develop sequential code, add parallel constructs and to debug concurrent code. The philosophy of phased migration (Figure 8) is very valuable when one is concerned about software modularity. Figure 9 elucidates the growth in the speed of computers over the last few decades together with projected growths. The graph clearly shows how the sequential processors have reached limits in speed and the advent of multiprocessors in the 1980's. The dotted line shows the past and projected growth in power on the small systems (FLEX/32) since its conception and indicates that such small parallel computers will soon compete with large computers in effective computing speed. It also shows the computational requirements in projected nonlinear dynamic response.

2.5 CONCLUDING COMMENTS

This section has provided a brief discussion of parallel processing issues and, an overview of selected parallel computer architecture. From these results one sees that future high speed computing is moving rapidly towards MIMD architectures as the most cost effective approach for providing high speed computing. Furthermore, these MIMD architectures are being incorporated into both supercomputers and moderate priced

computers to provide excellent capabilities for industrial and scientific applications. Parallel computers with 20-100 processors are now becoming commonplace and the 1990's will see 100-1000 processors in supercomputers and moderate priced systems. Cray, for example, is projected to have a 64 processor system by the early 1990's. Parallel architectures of the future are expected to be arrays of processors each containing vector capability. Memories will be large with various combinations of local and shared memories. The systems will be connected by a variety of communication systems such as bus, switch and rings. Some will have operating systems with the capability to independently program the individual processors, others will automatically map the tasks onto parallel processors according to what is allowed in the coding.

SECTION III
PARALLEL PROCESSING METHODS FOR NONLINEAR MSC/NASTRAN

3.1 INTRODUCTION

A recent study of parallel processing in MSC/NASTRAN (16) led to the development, testing and implementation of a parallel matrix decomposition in MSC/NASTRAN on the 4 processor CRAY X-MP and IBM 3090. To gain further insight into the potential of implementing MSC/NASTRAN on parallel computers this study investigates generation of nonlinear the force vectors stages. In the decomposition case the parallel code was readily incorporated largely because the subroutine interfaces and I/O were confined to narrow and well defined areas. For the force generation routines the interface and I/O interact through a wide variety of system routines. The Georgia Tech FLEX/32 computer with 8 processors was used as a development machine.

3.2. GENERATION OF NONLINEAR FORCE VECTOR IN MSC/NASTRAN

The generation of the nonlinear force vector is the key task of the nonlinear solution procedure in MSC/NASTRAN. This applies to nonlinear statics as well as to nonlinear transient dynamics. The force vector generation has to be performed for every iteration step regardless of the solution method employed. This operation is computationally expensive since all nonlinear elements have to be evaluated. Numerical experiments at MSC showed that for nonlinear problems of 2600 degrees of freedom (i.e. 900 QUAD 4 elements) more than 40% of the total CPU time may be required for nonlinear force vector calculations. For larger models this value might even exceed 50%.

Because of its importance and because of the inherent parallelism in its computations the corresponding MSC/NASTRAN module (NLEMG) was chosen as a candidate for parallel processing. The following gives a brief description of the NLEMG module with emphasis on the features which are critical for parallel processing. For simplification the discussion as well as the test problems are restricted to the case of geometric nonlinearity without temperature loads.

Figure 10 shows the basic double loop structure of the NLEMG module: composed of an outer loop over all element types and an inner loop over all elements of one type. It also shows the file reads which take place at different stages of program execution. In the geometric nonlinear case no file writes occur during the generation of the nonlinear force vector. Element type information is read from the ESTNL and ELDATA file. The element summary table and the nonlinear appendage data are read for each element (ESTNL data block). Finally the appropriate element routine (e.g. NQD4D) is called. Under the supervision of the element routine the element stiffness matrix is recovered from the KELM data block and immediately processed into an element force vector. During execution the ESTNL and ELDATA data blocks are processed in a specific order (via MSC/NASTRAN READ) while the element stiffness matrix is recovered in a random access type with FILPOS and GETSTR.

The layout of the MSC/NASTRAN open core memory prior to the element routine call is shown in Figure 11. Since the open core memory has to be reorganized for parallel processing, discussion of the current data structure in detail is necessary. Using pointers, the open core is divided into several sections, their length is determined prior to NLEMG (in

NLINIT). The use of these sections may be characterized by the following categories:

a) not used by NLEMG or accessed as read only, keeping certain global data in core (ICORE, LCORE, FCB-Blocks)

b) write out global results, nonlinear force vector (JCORE)

c) read and write element or element type specific data (EST-Appendage in ECORE, Thermal Data and Transformation Matrices in KCORE)

d) GINO I/O - Buffer

e) LEFT, not used at all

Besides open core a large number of common blocks are used. Most of them are unaltered during the NLEMG but some of them hold element type or element specific information. Certain common blocks are used as open core pointers. In general, common blocks can also be divided in classes of different use (see above), i.e. certain common variables stay unaltered during NLEMG, others may be updated for each new element type or they contain element specific data (e.g. /NLEST/ common block). The discrimination made above is of little interest for sequential program execution but it is critical for parallel processing.

3.3. GENERAL CONSIDERATIONS FOR PARALLEL PROCESSING

The generation of the nonlinear force vector has a high level of potential parallelism simply because it is mainly based on the evaluation of element data. In general, parallel processing could be invoked

a) on the element type level (element types are processed in parallel),

b) on the element level (elements of the same type are processed in parallel) or

c) by some kind of domain splitting (different portions of the model are processed in parallel).

Version (a) could be realized in MSC/NASTRAN by executing the outer loop of the NLEMG module in parallel but certain problems arise: the number of element types may be less than the number of available CPUs and it is very unlikely that the computational load among processes is evenly distributed. Version (b) involves parallel processing on a lower level of parallel granularity but since there are usually many elements in the model and elements of the same type are equally costly, load balancing problems are avoided.

Version (c) is another attractive possibility for parallel FEM analysis and it has been successfully implemented on a local memory machine (17) for linear statics and linear dynamics problems. However, problems arise especially for material nonlinear problems because it is not known a priori which portions of the model become nonlinear during the iteration. Certain domains may stay linear, others may be highly nonlinear and again computational load balancing becomes a problem. Furthermore, since the MSC/NASTRAN processing order is organized by element types and not by topology, major modifications of MSC/NASTRAN would be required.

Even on today's supercomputers with very large high speed memory, a complex nonlinear analysis requires the employment of secondary data storage devices (typically magnetic disks). Efficient disk input/output is a major feature of MSC/NASTRAN. With

the advent of parallel processing certain machine features have to be investigated in terms of disk input/output:

- Is the disk input/output possible from concurrent processes?
- Is the disk input/output handled exclusively through one node or an i/o controller?
- Are there several storage devices which can be accessed independently?

Since there is a very close coupling of disk input/output and computational work in the NLEMG module (see previous paragraph) it is crucial that the concurrent processes have equal accesses to the data base. Otherwise a major redesign of the code would be necessary to decouple disk input/output (performed sequentially) and "number crunching" (performed in parallel).[1]

3.4. DESCRIPTION OF THE PNLEMG PARALLEL MODULE

The parallel nonlinear force vector generation (PNLEMG) capability. As described earlier, the most practical approach to parallel processing is the concurrent evaluation of elements of the same type. To this end the current MSC/NASTRAN NLEMG module is interfaced before the element loop begins. If parallel processing is requested a new module named PNLEMG is called. This routine is a small driver to invoke parallel processing as shown in Figure 12.

---

1. Parallel code of this type has been successfully implemented in MSC/NASTRAN with the concurrent decomposition module PDCOMP (Ref. 16).

The first task of PNLEMG is the allocation of individual sections of open core memory for each process. This is necessary because each element has individual nonlinear appendage and temperature data. At the current stage of the project it is proposed that PNLEMG examines the amount of LEFT open core and establishes additional pointers for each process (Figure 13). This preliminary approach will work well for small and medium size geometrically nonlinear problems.[2] By choosing appropriate test problems this procedure is most practical to test PNLEMG in an application oriented environment without changing routines outside the current NLEMG module. At a later stage it should be feasible to allocate the required memory a priori and to modify the "spill feature" of the entire NLITER module accordingly.

The second task invokes parallel processing. This is realized by concurrent execution of the PNFVG ("parallel nonlinear force vector generation") routine. PNFVG essentially comprises the same code as the element loop of NLEMG, however, certain modifications are required.

As described above the nonlinear force vector generation in MSC/NASTRAN intertwines disk access and "number crunching" of in-core data. It is assumed that the available multiprocessor has only one channel to communicate with secondary storage but that all parallel processes have access to this device. This assumption reflects the current configuration of the GT FLEX/32 but it may or may not be true for different parallel

---

2. For the case "geometrically nonlinear only" the required memory for the nonlinear appendage is smaller than for material nonlinearities.

computer architectures. The incorporation of disk I/O into parallel processes certainly increases the program complexity and it may very well add additional machine dependencies compared to sequential MSC/NASTRAN code.

Assuming only one I/O channel for all processes simply means that only one of the parallel processes can access the secondary storage at a certain instant. However, this must not necessarily mean that only inferior parallel processing performance can be obtained. If the required I/O time is substantially less than the "truly parallel" executed code, each processor may use the I/O channel while the other processors carry out "in-core number crunching". Processes may efficiently run in a cascaded time schedule.

To implement a parallel processing nonlinear force vector without changing MSC/NASTRAN's database the following solution is proposed for PNFVG and all dependent routines (e.g. NLARED and the element routines). MSC/NASTRAN GINO routines are modified to be executable from parallel processes. However, all I/O functions must be protected from concurrent execution by special lock functions. If two or more processors should request a READ at the same time, one will wait until the other one is finished. As shown in Figure 14, I/O happens in PNFVG mainly at two stages, before the element routine is called (to retrieve EST and appendage data) and at the end of the element routine (to recover the element stiffness matrices).

This procedure is actually more restrictive than it would have to be. As long as the parallel processes receive their data directly from the buffer (which is located in the shared

memory region anyway) truly concurrent processing is possible. Only if the buffer is retrieved from the disk concurrent processing is not possible and locks have to employed. However, rather complicated interprocess communication and synchronization would have be incorporated on the GINO level. At the current stage a simpler solution is proposed.

The appendage data is stored in a reserved area of the open core memory (see above), the EST data is stored in the common block /ESTBUF/. This common block is defined local to each process.[3] The element stiffness matrix is read via the GINO buffer and immediately processed in an element force vector. No further storage of the matrix is necessary.

After the element force vector is obtained, the element contribution has to be assembled into the global nonlinear force vector. This vector is stored in shared open core memory but again, this operation is mutually exclusive for parallel processes. The locking procedure described above is used to assure that two parallel processes do not modify the same variables at the same time.

3.5. APPLICATION TO A TEST PROBLEM

The PNLEMG module was developed and tested on a FLEX/32 parallel computer at Georgia Tech in a standalone environment, i.e., the program was started from a special

---

3. In a multiprocessing environment "shared" and "local" common blocks may co-exist. Shared data is generally available to all parallel processes while local common blocks are available only to routines executing on the same processor.

MSC/NASTRAN data base dump made earlier at MSC. MSC/NASTRAN GINO calls were replaced by standard FORTRAN I/O. This procedure was necessary because MSC/NASTRAN is currently not implemented on the FLEX/32. The GINO modifications proposed above were not included in this project phase.

A geometrically nonlinear plate bending test problem consisting of 16 QUAD4 elements was chosen. The corresponding nonlinear force vector could successfully be generated on up to four parallel processors and the feasibility of the approach taken could be demonstrated.

The measured execution times for the force vector generation on 1,2,4 processors are given below:

- p=1 130 seconds
- p=2 124 seconds
- p=4 122 seconds

The results was obtained from parallel processing in the test problem show no significant speedup. Analysis of the results confirmed that the very limited speedup was caused by the extremely slow I/O from parallel processes on the FLEX/32. Transferring the same amount of data takes more than ten times longer from a parallel process than from sequential processing under UNIX. Therefore roughly 90% of the total execution time for the test problem goes into I/O and with only 10% of the time utilized for "number crunching". Since I/O is executed as a sequential task it is not surprising that the small speedup is was measured.

The extremely slow I/O capabilities of the FLEX/32 was discussed with the manufacturer where the I/O performance issues were confirmed. The total execution time may be broken up into $T_t = T_1 + T_2$, where T_1 is the time spent in physical I/O and wait for I/O access and T_2 is the execution time not related to I/O. Furthermore we can define a speedup related to the time T_2 (which is the actual parallel part) as

$$S_2 = \frac{T_2 \text{ on n processors}}{T_2 \text{ on 1 processor}}$$

The following results are obtained

Number of processors	T_1	T_2	S_2
1	117	13	1.0
2	117	7	1.8
4	117	5	2.6

With respect to the relatively small problem of only 16 elements the results obtained show a satisfactory decrease of computing time due to parallel processing.

However, more important than the speedup figures is the validity of the overall concept and the proper functioning of the developed code. Both of these were demonstrated sufficient for this initial stage of program development. The actual benefits of parallel processing for nonlinear FEM analysis remain to be investigated in an actual Nastran environment on a machine with fast I/O capabilities from parallel processes.

3.6. CONCLUSIONS

A new program module to incorporate parallel processing into MSC/NASTRAN's nonlinear force vector generation has been developed and tested. Elements of the same type are processed concurrently under the supervision of a parallel processing driver routine. The current MSC/NASTRAN data base is used and the module can be easily incorporated in the nonlinear solution sequences. At this stage the application is limited to geometrically nonlinear analysis and QUAD4 elements but the module can be upgraded to incorporate all features of MSC/NASTRAN's nonlinear analysis.

Computer hardware considered for implementation has to provide efficient access to secondary storage devices from parallel processes. The current program version utilizes input/output functions from parallel processes but makes this task mutually exclusive. Therefore the data base is processed by one processor only at any instant.

Initial tests on the FLEX/32 proved the feasibility of the concept but the actual benefits of parallel in a production environment processing for nonlinear analysis remain to be investigated.

3.7. FUTURE WORK

The results of this study show the feasibility of parallel processing within MSC/NASTRAN's nonlinear solution sequence. The developed code was transferred to MSC and implemented in solution 66 as a part of a summer internship at MSC by Dietmar Goehlich August/September 1987. The implementation progressed satisfactorily on a DEC VA7 8700 but was delayed due to DEC operating system issues uncovered by the

implementation. Several issues still remain to be investigated and further code development is required before a parallel force vector generation can be installed in a production MSC/NASTRAN. Followon tasks along one of the following two directions appear appropriate.

1. Further Development of Parallel MSC/NASTRAN Source Code.

   Work proposed in this context is a direct continuation of the PNLEMG project reported here in. The general strategy is to map computationally expensive tasks across parallel processes but execute the majority of routines sequentially. This approach is meaningful for approximately 2-8 parallel processes but may not be prudent for highly parallel environments (e.g. machines with 64 or more cpus) because a significant portion of the code (approximately 5-10%) remains serial. Possible projects include the following:

   - Testing the PNLEMG module in an application oriented environment. Assessment of execution time gain and computational efficiency for several test problems and iteration strategies.
   - Enhancement of the PNLEMG module to incorporate the full set of nonlinear elements including material nonlinearities and temperature loads.
   - Parallel generation and assembly of global matrices (stiffness, mass, damping).

2. Study on System Design for Parallel Processing

The need for a new overall code design may arise at the end of this decade because of the advent of massively parallel computers and the demand for increasing computing power. But the employment of 32,64 or more parallel processors is only meaningful if almost the entire code (more than 95% of the cpu time!) is executed in parallel. Since the current approach to parallel processing is module oriented and only computational expensive tasks are mapped across parallel processes it may not be possible to reach this level of parallelism.

As pointed out earlier the finite element approach can be organized in such a way that certain partitions of the model are processed independently on parallel processes and only the information on common boundaries is shared between processes. In principle this procedure resembles the substructuring or superelement approach which is already implemented in MSC/NASTRAN. Substructuring in the strict sense is limited to linear analysis. For nonlinear analysis some kind of domain splitting has to be employed. Algorithms which balance the computational load among parallel processors and sophisticated interprocess communication schemes have to be devised.

By the virtue of this approach theoretically the entire code may be executed in parallel, input/output functions could run on independent

secondary storage devices. However, proper load balancing and an appropriate data base design are not easily achieved especially for nonlinear problems.

A simple prototype code based on current Nastran capabilities could be developed to study the feasibility and required features of such a highly concurrent FEM code.

Further work is necessary to incorporate parallel computing in the processing of the MSC/NASTRAN data base. The new Executive system with the RAM disk and buffer pooling capabilities may be effectively exploited for parallel processing.

SECTION IV

INVESTIGATION OF PARALLEL NONLINEAR DYNAMIC ANALYSIS METHODS

4.1. INTRODUCTION

This section reports on an investigation of several algorithms to be used as the basis for integration of large order coupled dynamic equations. The area studied include integration methods, approaches to solving simultaneous equations and related issues.

The investigation first examines the parallel implementation of the cyclic reduction method for solving sets of equations. The comparison of this method with LDL^T decomposition method is made. The cyclic reduction method is also incorporated in the Newmark-β method to solve a nonlinear dynamic problem. The central difference method for parallel time integration is also investigated. When the central difference method is used in the case of nondiagonal mass matrix, several iterative methods (i.e. the Jacobi method, the mixed Jacobi/Gauss-Seidel method and the conjugate gradient method) are examined and compared.

4.2. FORMULATION OF PROBLEM

The equations of motion for a nonlinear time varying dynamic system can be expressed in the form

$$\mathbf{M\ddot{u} + C\dot{u} + Ku = f} \qquad (1)$$

where M, C, K are the mass, damping and stiffness matrices of the system, and **f** is the external force vector. The variables $\ddot{\mathbf{u}}$, $\dot{\mathbf{u}}$, **u** correspond to acceleration, velocity, and displacement vectors of the system. Since the system is nonlinear, M, C, and K are not

necessarily constants and may require updating to current values at every integration step.

There are numerous methods in the literature for integrating nonlinear dynamic equations over time. Alternate integration strategies include implicit versus explicit methods. Typical methods used in commercial finite element codes include the Newmark (18), Houbolt (19), Wilson (20) and central difference method (21). Table 4.1 lists several implicit and explicit integration methods and Table 4.2 gives selected details relative to their use for integration of nonlinear equations on parallel computers. These initial studies suggest that the central difference and Newmark methods are suitable methods for parallel implementation. Each has certain parallel deficiences. For example, the central difference method is best suited when the mass matrix is diagonal and the Newmark method requires use of the matrix decomposition. The present study investigates ways to improve each of the two methods for parallel implementation.

4.3 DESCRIPTION OF TEST PROBLEM

Transient response problem typical that shown in Figure 15 is used in the study as the test problem at the first stage of research. In Figure 15, S_{n1}, S_{n2},..., S_{nn} are nonlinear springs where the spring force is proportional to the cubic of the displacements. The test problem leads to tridiagonal matrices which are subsequently modified in the study to have an increasing number of diagonals so that the effect of matrix sparsity can be studied.

The parallel algorithms studied are implemented on a FLEX/32 8 processor multicomputer at Georgia Tech with a shared memory capability (11) (See Figure 4).

4.4. THE CYCLIC REDUCTION METHOD

When some implicit integration methods such as the Newmark-β method are used to solve Equation 1, and when some explicit methods such as the central difference method are used in the case of nondiagonal mass and/or damping matrix, sets of algebraic equations are produced and must be solved. Therefore, solving sets of simultaneous equations can be a significant part of the nonlinear dynamic solution and some related techniques should be studied from the view point of parallel nonlinear dynamic computation. Recently, the parallel implementation of the LDL^T decomposition method has been investigated by several researchers (22-24). The results of research (16) show that the parallel implementation of LDL^T decomposition method can be quite efficient if the half bandwidth is large enough. However, for small bandwidth problems, the parallel Cholesky is inefficient. Therefore, it is appropriate to investigate another parallel method that is efficient for small bandwidth problem. As a candidate of the expected method, the cyclic reduction method is examined first in the present study.

4.4.1. Literature Survey for Cyclic Reduction

The cyclic reduction method was used by Hockney as early as in 1965 (25). He used it to solve tridiagonal systems arising from the finite difference approximation to Poisson's equation. He chose the method in preference to Gaussian elimination

because cyclic reduction deals with periodic boundary conditions without the need for the calculation of auxiliary vectors. For convenience, the number of mesh points and therefore the number of equations was taken to be a power of two.

Later, the method was also used by Buneman (26), Busbee, Golub and Nielson (27). Busbee, Golub and Nielson showed how to generalize this method to apply to more general elliptic partial differential equations, different types of boundary conditions, and more general rectilinear domains. Dorr described the work above in his survey paper (28) and noted that there appears to be some degree of computational instability when the algorithm is implemented in exactly the way presented by Buzbee, Golub and Nielson. Dorr also pointed out that Buneman had devised a similar procedure that is computationally stable (26).

At the early stage of use of cyclic reduction the basic drawback of the method was a restriction on the block size of the matrix, i.e., it must be a power of two. Sweet presented a generalization of cyclic reduction process to an arbitrary block size for the discrete Poisson's equation (29). Later, he gave a generalization of the Buneman variant of cyclic odd-even reduction algorithm for solving finite difference approximation to Poisson's equations. This generalization also places no restriction on the block size of the system. Swarztrauber (30) extended the cyclic reduction method to linear systems which result from the discretization of superable elliptic equations with Dirichlet, Neumann or periodic boundary conditions. The algorithm presented is also free from the

restriction for the number of equations.

In 1976, Heller presented his paper (31) in which the solution of general block tridiagonal linear systems by a cyclic odd-even reduction algorithm is considered. He pointed out that under conditions of diagonal dominance, norms describing the off-diagonal blocks relative to the diagonal blocks decrease quadratically with each reduction. This may allow early termination of the reduction when an approximation solution is desired. Later, Sweet gave also a cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension (32).

To examine the performance of cyclic reduction in a parallel environment Stone (33) made a comparison of the cyclic odd-even reduction algorithm with the recursive doubling algorithm and an algorithm based on Buneman's Poisson solver. He concluded that for pipeline computers similar to CDC STAR, the cyclic odd-even reduction algorithm appears to be the most preferable algorithm for all cases. Another job in this area was done by Lambiotte and Voigt (34). They gave program timings for the CDC STAR comparing odd-even reduction, sequential Gaussian elimination, and a consistent variation of Stone's method. Madsen and Rodrigue (35) also discussed a program for the STAR in which a polyalgorithm was created by using odd-even reduction until reduced system is obtained where some other method is faster. They use the LU factorization, odd-even elimination and odd-even reduction for small, moderate and large number of equation respectively. This idea is also discussed in the book by

Hockney and Josshope (36). In the book Hockney and Josshope also mentioned the possibility of terminating the reduction prior to the completion of the whole process if the dominance of the system is strong enough and approximate solutions are desired.

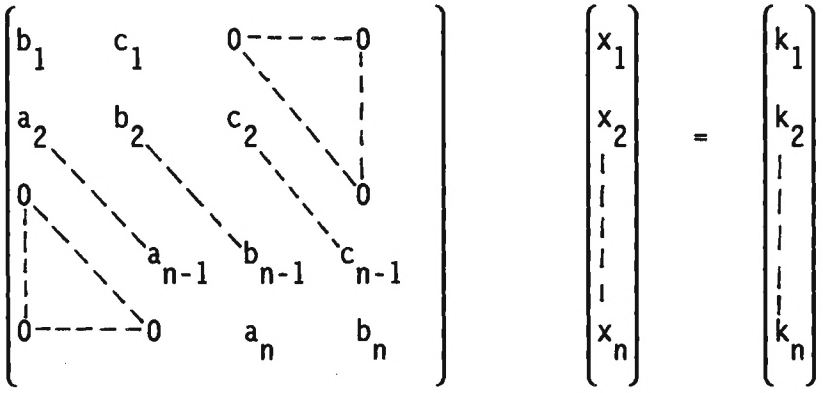
The parallel implementation of cyclic reduction for block tridiagonal matrices was studied by Kapur and Browne (37,38). They did the work on the TRAC experimental parallel computer at the University of Texas. They also considered the cyclic elimination method which was superior to cyclic reduction on the TRAC. The reason for this is that the extra operations of the elimination method are done in parallel at no extra cost and because there is no back substitution step.

Another study of parallel cyclic reduction is made by Opsahl and Parkinson (39). They described an algorithm of cyclic reduction for solving almost tridiagonal problems on SIMD computers like the DAP and MPP. The algorithm exploits the sparsity structure of this class of problems, thereby allowing parallel cyclic reduction to be used as the main step in the solution procedure.

4.4.2. Cyclic Reduction Algorithm

The cyclic reduction method is an elimination method for tridiagonal type equations where the reduction strategy is based on eliminating every other variable and equation. These eliminations can be assigned to independent processors to exploit parallelism.

The general tridiagonal set of linear algebraic equations may be written as:



or in matrix-vector notation:

$$\mathbf{Ax} = \mathbf{k}$$

For simplicity the number of equation, n, is taken to be a power of two. As mentioned in the literature survey, however, the restriction is not a necessity of the method. Assuming that n $= n' - 1$, where $n' = 2^q$ and q is an integer, the cyclic reduction algorithm may be stated as follows (20).

For levels $\ell = 1,2,...,q-1$, perform the recursive calculation of new coefficients and right-hand sides from

$$a_i^{(\ell)} = \alpha_i \, a_{i-2}^{(\ell-1)}$$

$$c_i^{(\ell)} = \gamma_i \, c_{i+2}^{(\ell-1)}$$

$$b_i^{(\ell)} = b_i^{(\ell-1)} + \alpha_i c_{i-2}^{(\ell-1)} + \gamma_i a_{i+2}^{(\ell-1)}$$

$$k_i^{(\ell)} = k_i^{(\ell-1)} + \alpha_i k_{i-2^{(\ell-1)}}^{(\ell-1)} + \gamma_i k_{i+2^{(\ell-1)}}^{(\ell-1)}$$

where

$$\alpha_i = - a_i^{(\ell-1)} / b_{i-2^{(\ell-1)}}^{(\ell-1)}$$

$$\gamma_i = - c_i^{(\ell-1)} / b_{i+2^{(\ell-1)}}^{(\ell-1)}$$

and

$$i = 2^\ell \text{ step } 2^\ell \text{ until } n' - 2^\ell ,$$

with the initial values $a_i^{(0)} = a_i$, $b_i^{(0)} = b_i$, $c_i^{(0)} = c_i$ and $k_i^{(0)} = k_i$. Then, for $\ell = q, q-1,...,2,1$ perform the back substitution from

$$x_i = (k_i^{(\ell-1)} - a_i^{(\ell-1)} x_{i-2^{(\ell-1)}} - c_i^{(\ell-1)} x_{i+2^{(\ell-1)}}) / b_i^{(\ell-1)}$$

where

$$i = 2^{(\ell-1)} \text{ step } 2^{(\ell-1)} \text{ until } n'-2^{(\ell-1)}$$

and $x_0 = x_{n'} = 0$ when they occur.

In the cases of five or more diagonal system, the concept of block tridiagonal system is used, which means that a_i, b_i, c_i, α_i, γ_i are matrices and k_i, x_i are vectors instead of scalar numbers.

4.4.3 Comparison of Cyclic Reduction with the LDL^T Decomposition Method

Figure 16 compares the LDL^T decomposition and cyclic reduction methods as a function of bandwidth and number of processors. In the figure relative speedup is defined as the execution time of the LDL^T decomposition method on one processor divided by the execution times of the cyclic reduction method on n processors. Figure 16 illustrates that a

parallel LDL^T decomposition is not effective for solution of small bandwidth problems. In contrast, in the case of cyclic reduction, the speedup is quite good for small bandwidth problems. Particularly in the case where the half bandwidth is 2 or 3, the cyclic reduction method gives much higher efficiency than LDL^T decomposition method in parallel environment. Another noticeable fact shown by Figure 16 is that the speed of the cyclic reduction method on one processor is much slower than that of the sequential LDL^T decomposition method when the half bandwidth is 5 or greater. Therefore, even the use of 7 processor cannot make the cyclic reduction method faster than the sequential implementation of LDL^T decomposition method.

Another way to compare the two methods is shown in Figure 17 where the relative processor utilization is defined as the ratio of relative speedup to number of processors. Figure 17 shows that the relative processor utilization in the case of cyclic reduction drops off rapidly from high values to about 10% as the half bandwidth is increased from 2 to 5. This means that the total set of processors are not being used efficiently. In contrast, the processor utilization in the case of LDL^T decomposition method increases as the half bandwidth increases. From Figure 16 and Figure 17 it can be seen that the cyclic reduction method is much more efficient than the LDL^T decomposition method in parallel implementations where the half bandwidth is small (i.e. 2 or 3). But in the case of a larger half bandwidth, say 5 or larger, the LDL^T decomposition method would be better than the cyclic reduction method. To verify the performance of cyclic reduction shown in the

numerical experiments the numbers of operation involved in the cyclic reduction for cases of several half bandwidths are calculated (See Figure 18). In Figure 18 the relative number of operation is defined as the ratio of number of operation for some half bandwidth to the number of operation for tridiagonal system. Similarly , the relative executive time is defined as the ratio of execution time for some half bandwidth to the execution time for tridiagonal system. From Figure 18 it can be seen that the variation trends of the relative execution time are close to that of the relative number of operation, which validates the performance of cyclic reduction obtained by the numerical experiments.

Although the results indicate that the parallel cyclic reduction is worse than the LDL^T decomposition method when the half bandwidth is greater than a value, the bandwidth can be enlarged when the number of parallel processors is large. Moreover, if the diagonal dominance of the system is strong and some kind of approximation results are acceptable, the cyclic reduction process can be terminated earlier and the number of operation required can be decreased. This also can enlarge the range of half bandwidth in which cyclic reduction is more efficient than the LDL^T decomposition method.

Considering that the parallel LDL^T decomposition method begins to be inefficient when the half bandwidth is less than about 50 (16), there exist a range of bandwidth (say, from 5 to 50) in which neither the cyclic reduction nor the LDL^T decomposition methods are efficient from the standpoint of parallel computation. Another more efficient method should be investigated for this intermediate range and further study is needed.

4.5 THE NEWMARK-β METHOD USING CYCLIC REDUCTION

As a part of the study of integration methods the Newmark-β method is implemented in parallel and the cyclic reduction is incorporated for the purpose of examining its use in dynamic problems. The Newmark-β method is based on the following approximation:

$$\dot{u}_{t+\Delta t} = \frac{\alpha}{\beta \Delta t}\left(u_{t+\Delta t} - u_t \right) - \left(\frac{\alpha}{\beta} - 1 \right) \dot{u}_t - \Delta t \left(\frac{\alpha}{2\beta} - 1 \right) \ddot{u}_t \tag{2}$$

$$\ddot{u}_{t+\Delta t} = \frac{1}{\beta \Delta t^2}\left(u_{t+\Delta t} - u_t \right) - \frac{1}{\beta \Delta t} \dot{u}_t - \left(\frac{1}{2\beta} - 1 \right) \ddot{u}_t \tag{3}$$

Substituting Equations (2) and (3) into (1) yields

$$\bar{M}\, u_{t+\Delta t} = \bar{f}_{t+\Delta t} \tag{4}$$

where

$$\bar{M} = \frac{1}{\beta \Delta t^2} M + \frac{\alpha}{\beta \Delta t} C + K$$

$$\bar{f}_{t+\Delta t} = f_{t+\Delta t} + \left[\left(\frac{1}{2\beta} - 1 \right) M + \Delta t \left(\frac{\alpha}{2\beta} - 1 \right) C \right] \ddot{u}_t + \left[\frac{1}{\beta \Delta t} M + \left(\frac{\alpha}{\beta} - 1 \right) C \right] \dot{u}_t + \left[\frac{1}{\beta \Delta t^2} M + \frac{\alpha}{\beta \Delta t} C \right] u_t$$

Since $\bar{M}$ is nondiagonal a set of equations must be solved at each time step.

Figure 19 shows the results of computational speedup versus number of processors for the test problem in which $\bar{M}$ is a tridiagonal matrix and Equation (4) is solved by cyclic reduction. Here speedup is defined as the time taken to solve a problem on one processor divided by the time to solve the same problem on n processors. Thus, the speedup on one processor is one and the theoretical maximum speedup (with no overhead) on n processor is n. Figure 19 shows that the computation speedup is degraded for small degrees of freedom but is quite good as the number of degrees of freedom increases. The improvement of speedup with large degree of freedom is due to the relatively small importance of such overhead factors as communication, lack of synchronization, and unequal distribution of computation tasks.

4.6. THE CENTRAL DIFFERENCE METHOD

The central difference method is also one of the integration methods which are of interest. The approximation used in this method is as follows

$$\dot{u}_t = \frac{1}{2\Delta t}\left[u_{t+\Delta t} - u_{t-\Delta t}\right] \tag{5}$$

$$\ddot{u}_t = \frac{1}{\Delta t^2}\left[u_{t+\Delta t} - 2u_t + u_{t-\Delta t}\right] \tag{6}$$

Substituting (5) and (6) into (1) gives

$$\bar{M}\, u_{t+\Delta t} = \bar{f}_t \tag{7}$$

where

$$\bar{M} = \frac{1}{\Delta t^2} M + \frac{1}{2\Delta t} C$$

$$\bar{f}_t = f_t - \left(K - \frac{2}{\Delta t^2} M\right) u_t - \left(\frac{1}{\Delta t^2} M - \frac{1}{2\Delta t} C\right) u_{t-\Delta t}$$

In the case of diagonal mass matrix and diagonal damping matrix, the effective matrix $\bar{M}$ in Equation (7) is diagonal. Therefore, the solution can be obtained directly from Equation (7) and the parallelism is quite good. Figure 20 shows results of computation speedup versus number of processors for several large degrees of freedom examples.

For the case where the mass matrix and/or damping matrix are nondiagonal Equation (7) becomes a coupled set of equations and the central difference method requires the solution of a set of simultaneous equations. Figure 22 shows the results in such a case where a mixed Jacobi/Gauss-Seidel iteration method is used to solve the equations. The method is Gauss-Seidel within each processor and Jacobi between processors. For small numbers of processors it has the efficiency of the Gauss-Seidel while retaining the parallel benefits of Jacobi. Figure 21 shows the logic of the method and processor assignment. The computation results are shown in Figure 22 from which it can be seen that the mixed Jacobi/Gauss-Seidel method is more efficient than the Jacobi in parallel environment.

Since iterative methods have good parallelism and are attractive for parallel computation, it is in order to try to let it be more efficient as incorporated in integration

scheme. Using extrapolation to get a better starting value at each time step can help to fulfill the task. Figure 23 shows that extrapolation can make the algorithm significantly more efficient. Here the Newton-Gregory forward polynomial is used to perform the extrapolation.

Also because of the good parallelism and attractiveness of iterative methods it is necessary to study several iterative schemes more from the view point of parallel nonlinear dynamic computation. Several researchers have done some work (40-45). As the first step of the research in this direction a comparison of the mixed Jacobi/Gauss-Seidel with the conjugate gradient method is made (See Figure 24). Figure 24 indicates that the mixed Jacobi/Gauss-Seidel method is more efficient than the conjugate gradient method when the value of diagonal element is equal to 4. As the value of diagonal element decreases to 3.05 the efficiency of mixed Jacobi/Gauss-Seidel method is severely degraded and approaches instability while the conjugate gradient method is still efficient and stable. The results suggest that the conjugate gradient method is more stable for small value of diagonal element but less efficient than the mixed Jacobi/Gauss-Seidel method when the value of diagonal element is large enough.

4.7 CONCLUDING REMARKS

The parallel implementation of cyclic reduction is performed and its comparison with the LDL^T decomposition method is made. The results indicate that the cyclic reduction method is much more efficient than the LDL^T decomposition in parallel implementation when the half bandwidth is small (i.e. 2 or 3). As the half bandwidth is greater, however, the

parallel cyclic reduction is slower than sequential LDL^T decomposition method if the number of processor is limited (say, 7 or less) because the number of operation is increased significantly. But the range of half bandwidth in which parallel cyclic reduction is more efficient than the LDL^T decomposition method can be enlarged in the following cases.

1. When the number of processors is greater.
2. When the diagonal dominance of the system is strong enough and some kind of approximation results are acceptable.

As for the study of integration methods, the Newmark-β method incorporating cyclic reduction and the central difference method for the cases of diagonal and nondiagonal mass matrices are implemented parallelly. In the case of nondiagonal mass matrix, some iterative schemes such as Jacobi and mixed Jacobi/Gauss-Seidel methods are used. To improve the efficiency of iteration the extrapolation technique is applied and good results are obtained. The initial results suggest that iterative methods are attractive for dynamic problems with a nondiagonal mass matrix. As a preliminary study of iterative methods the comparison of mixed Jacobi/Gauss-Seidel method with the conjugate gradient method is made. All the initial results are encouraging and suggest that further detailed studies be carried out.

OVERALL CONCLUDING REMARKS

This report summarizes an investigation and evaluation of selected algorithms for MSC/NASTRAN which appear well suited for nonlinear dynamic analysis on parallel super computers. The study focused on several areas including:

1. Review of architectural chracteristics of parallel computers relative to finite element methods. The study characterized several architecture and identified key characteristics needed for finite element methods. By 1990 numerous parallel processing computers will be widely available.
2. Redesign and implementation of specific MSC/NASTRAN subsystems on a parallel computer to assess parallel processing issues and I/O characteristics. Test bed software used in the study was portions of the nonlinear MSC/NASTRAN force vector computations used for nonlinear analysis of test problems. The study demonstrated an approach for parallel computation of MSC/NASTRAN nonlinear force vectors.
3. Investigation of candidate methods for nonlinear dynamic analysis on a parallel computer. Areas studied included a parallel implementation of the cyclic reduction method for sparce matrices. The modification to facilitate parallel computations was achieved for the central difference method for nondiagonal mass matrices and the Newmark method to facilitate parallel computations. The study showed that the cyclic reduction method is superior to decomposition methods for highly sparce systems. Modifications to central difference and

Newmark methods are shown which make them better suited for parallel computing.

It is recommended that this research toward a parallel MSC/NASTRAN be continued and that the next effort focus on the following tasks.

a. Investigation of appropriate program designs and data structures for vector and parallel processing.

b. Investigation of decoupled parallel numerical integration approaches for nonlinear dynamics.

c. Investigation of data management requirements for MSC/NASTRAN to support parallel finite element computations.

The results of these research tasks should lead to the following results:

a. Description of a program system architecture and appropriate data structure to perform highly efficient finite element analysis on vector and parallel computers.

b. Recommendation of a parallel integration method for nonlinear dynamics.

c. An assessment of key parallel data management issues in MSC/NASTRAN.

REFERENCES

1. Butler, T. and Michael, D.: NASTRAN, A Summary of Functions and Capabilities of the NASA Structural Analysis Computer System, NASA SP 260, 1971.

2. Fredriksson, B., and Mackerle, J.: Partial List of Major Finite Element Programs and Description of Some of Their Capabilities. In State-of-the-Art Surveys on Finite Element Technology (Noor and Pilkey, Editors). ASME Publication H00290, 1983, pp. 363-403.

3. Noor, A., Storaasli, O., and Fulton, R.: Finite Element Technology in the Future. Impact of New Computations on Computational Mechanics (Noor and Pilkey, Editors), ASME Special Publication H--275, November 1983, pp. 1-32.

4. Adeli, H. and Vishnubhotla, P., "Parallel Processing, Microcomputers in Civil Engineering", Vol. 2, No. 3, Sept. 1987.

5. Smarr, L. L., "The Computational Science Revolution: Technology, Methodology and Scoiology", in High Speed Computing: Scientific Applications and Algorithm Design, Ed. by R. B. Wilhelmson, University of Illinois Press, 1987.

6. McCormick, B. H., DeFanti, J. A. and Brown, M. D., "Visualization in Scientific Computing", Report to NSF by Panel on Graphics, Image Processing and Workstations, July 1987, See also IEEE Computer Graphics and Applications, July 1987, pp. 61-70.

7. Noor, A. K., Storaasli, O. O. and Fulton, R. E. "FEM Hardware and Postprocessing", Finite Element Handbook (Kardestuneer and Norrie, Editors), pp. 4.209-4.3 .2, McGraw-Hill, 1987.

8. Chandler, D., "The Business of Parallel Processing in Business", UNIX Review, 5(6): 17-27, 1987.

9. Encore, Multimax Technical Summary, Encore Computer Corporation, Malboro, Massachusetts, 1985.

10. "FLEX/32 Multicomputer-System Overview", Document 030-000-001, Flexible Computer Corporation, Dallas, Texas, 1985.

11. Matelan, N., "The FLEX/32 Multicomputing Environment", in Research in Structures and Dynamics - 1984, R. J. Hayduk and A. K. Noor (compilers), NASA CP 2335, 1984.

12. IPSC, "iPSC System Overview Manual:, Intel Corporation, 1986.

13. Pountain, D., "Microprocessor Design", Byte, August 1984, pp. 361-365.

14. Walker, P., "The Transputer - A Building Block for Parallel Machine", Proceedings of the Ninth Conference on Electronic Computation, American Society of Civil Engineers, New York, 1986, pp. 692-700.

15. PIXAR Reference Manual, PIXAR, 1987.

16. Goehlich, D., Komzsik, L., and Fulton, R. E., "Decomposition of Finite Element Matrices on Parallel Computers", ASME Computers in Engineering Conference, August 1987, New York, NY.

17. C. Farhat, "Multiprocessors in Computational Mechanics", Ph.D. Dissertation, University of Clifornia, Berkeley, CA, 1986.

18. Newmark, N., "A Method of Computation for Structural Dynamics", J. Engr. Mech. Div., ASCE, July 1959, EM3, pp. 67-94.

19. Houbolt, J. C., "A Recurrence-Matrix Solution for the Dynamic Response of Elastic Aircraft," NASA TN 2060, 1950.

20. Wilson, E. L., "A Computer Program for the Dynamic Stress Analysis of Underground STructures," Report No. SESM 68-1, Department of Civil Engineering, University of California, Berkeley, 1968.

21. Collatz, L., "The Numerical Treatment of Differential Equations," Springer-Verlag, New York, 1966.

22. Chen, S. S., and Dongarra, J. J., "Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences with Small Granularity", Journal of Parallel and Distributed Computer, 1, 1984, pp. 22-31.

23. George, A., Heath, M. T., and Liu, J., "Parallel Cholesky Factorization on a Shared-Memory Multiprocessor", Tech Rept. ORNL-6124, March 1985, Oak Ridge National Laboratory.

24. Heath, M. T., "Parallel Cholesky Factorization in Message-Passing Multiprocessor Environments," Tech. Rept. ORNL-6150, March 1985, Oak Ridge National Laboratory.

25. Hockney, R. W., "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis," J. Assoc. Comp. Mach. 12, 95-113.

26. Buneman, O., "A Compact Non-Iterative Poisson Solver", Rep. SU-IPR-294, Institute for Plasma Research, Stanford University, Stanford, California, 1969.

27. Buzbee, L., Golub, G. H., Nielson, C. W., "The Method of Odd-Even Reduction and Factorization with Application to Poisson's Equation", Rep. LA-4141, Los Alamos Scientific Laboratory, Los Alamos, New Mexico, 1969.

28. Door, F. W., "The Direct Solution of the Discrete Poisson EQuation on a Rectangle", SIAM REview 12, 248-263, 1970.

29. Sweet, R. A., "A Generalized Cyclic Reduction Algorithm" SIAM J. Num. Anal. 11, 506-20, 1974.

30. Swarztrauber, P. N., "The Method of Cyclic Reduction, Fourier Analysis and the FACR Algorithm for the Discrete Solution of Poisson's Equation on a Rectangle", SIAM Review, Vol. 19, No. 3, July 1977.

31. Heller, D., "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems", SIAM J. Num. Anal. 13, 484-496.

32. Sweet, R. A., "A Cyclic Reduction Algorithm for Solving Block Tridiagonal System of Arbitrary Dimension", SIAM J. Num. Anal. 14, 706-20, 1977.

33. Stone, H. S., "Parallel Tridiagonal Equation Solvers", ACM Transactions on Math. Software, 1, No. 4, 289-307, December 1975.

34. Lambiotte, J., and Voigt, R. G., "The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer", ACM Trans. Math. Software, 1, 1975, pp. 308-329.

35. Madsen, N. K., and Rodrigue, G. H., "A Comparison of Direct Methods for Tridiagonal Systems on the CDC STAR-100", Lawrence Livermore Lab., Livermore, CA, 1975.

36. Hockney, R. W., and Jesshope, C. R., "Parallel Computers", Adam Hilger Ltd., Bristol, 1981.

37. Kapur, R., and Browne, J., "Block Tridiagonal Linear Systems on a Reconfigurable Array Computer", Proc. 1981 Int. Conf. Par. Proc., pp. 92-99, 1981.

38. Kapur, R. W., and Browne, J., "Technique for Solving Block Tridiagonal Systems on Reconfigurable Array Computers", SIAM J. Sci. Stat. Comp., 5, pp. 701-719, 1984.

39. Opsahl, T., "An Algorithm for Solving Sparse Sets of Linear Equations with an Almost Tri-diagonal Structure on SIMD Computers", Proc. 1986 Int. Conf. Parallel Processing, pp. 369-374.

40. Adams, L., and Ortega, J., "A Multicolor SOR Method for Parallel Computation", Proc. 1982 Int. Conf. Parallel Processing, pp. 53-56.

41. Adams, L., and Jordon, H., "Is SOR Color-Blind? ICASE Report 84-14, NASA-Langley Research Center, 1984.

42. Johnson, O., Micchelli, C., and Paul G., "Polynomial Preconditioners for Conjugate Gradient Calculations", SIAM J. Num. Anal. 20, 1983, pp. 362-376.

43. Adams, L. "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers", Ph.D. Thesis, University of Virginia, 1982.

44. Adams, L., "An M-Step Preconditioned Conjugate Gradient Method for Parallel Computation", Proc. 1983 Int. Conf. Parallel Processing, pp. 36-43.

45. Schreiber, R., and Tang. W., "Vectorizing the Conjugate Gradient Method", Proc. Symposium Cyber 200 Applications", Ft. Collins Co., 1982.

(I) **Explicit Method** : Can take advantage of lumped mass, suitable for local memory and shared memory machine

$$M\Delta\ddot{u}_t = \Delta Q_t - K\Delta u_t - C\Delta\dot{u}_t$$

(a) Central Difference Method

(b) Two-Cycle Iteration with Trapezoidal Rule

(c) Runge-Kutta Methods

(II) **Implicit Method** : Can take advantage of large Δt , not suitable for local memory machine, since local memory machine such as the iPSC Hypercube has difficulty to handle matrix decomposition.

$$\hat{K}u_t = \hat{Q}_t$$

(a) Newmark Beta Method

(b) Houbolt Method

(c) Wilson Theta Method

(d) Park Stiffly Stable Method

Table 4.1. Types of Numerical Integration Methods.

(1) Newmark Method

- Implicit method
- unconditional Stable ($\alpha = 1/2, \beta = 1/4$)
- Suitable for nonlinear dynamics analyses.
- LDL^T type
- NR family
- Speedup potential is depends on matrix decomposition

(2) Central Difference

- Explicit Method
- May go unstable
- Suitable for nonlinear dynamics analyses.
- Marching forward
- Very good speedup potential even for large number of processors

(3) Two-Cycle Iteration

- Explicit Method
- May go unstable (worse than Central Diff.)
- Marching forward
- Very good speedup potential even for large number of processors

(4) Houbolt Method

- Implicit Method
- Stable
- Non-Self-Starting
- Requires store disp. for three previous time steps
- Can use of cubic extrapolation
- Suitable for nonlinear dynamics analyses
- LDL^T type
- NR family
- Speedup potential is depends on matrix decomposition

Table 4.2. Characteristics of Integration Methods for Parallel Computers.

(5) Runge-Kutta

- Explicit Method
- Highly numerical damping
- Each forward step requires several evaluations of the function
- Marching forward
- Speedup potential is high

(6) Park Stiffly

- Implicit Method
- Stable
- Non-Self-Starting
- Requires store disp. & veloc. for three previous time steps.
- LDL^T type
- NR family
- speedup potential is depends on matrix decomposition

(7) Wilson Theta

- Implicit Method
- Stable
- LDL^T type
- NR family
- Speedup potential is depends on matrix decomposition

Table 4.2. (Continued)

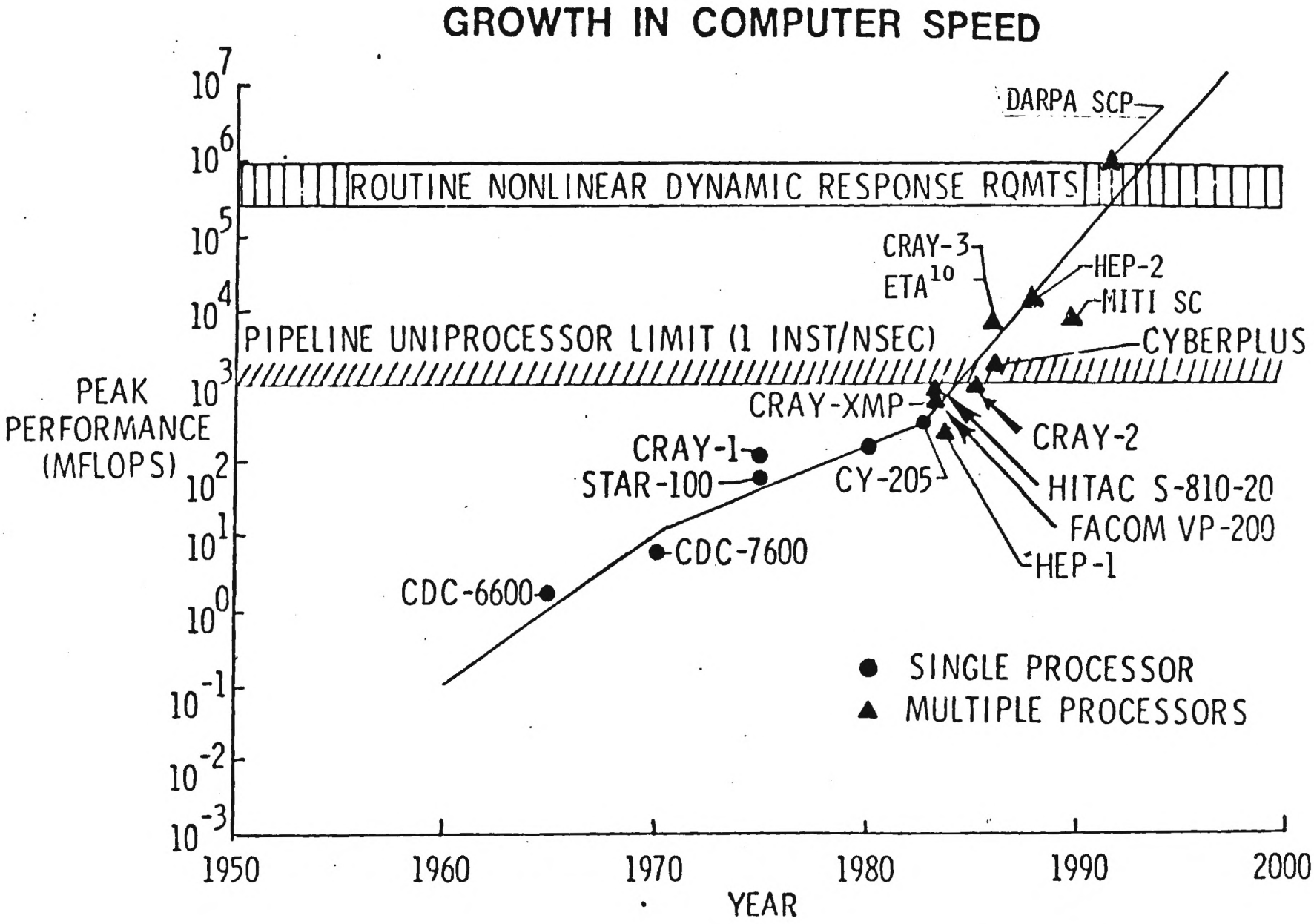


Figure 1. Growth in Computer Speed.

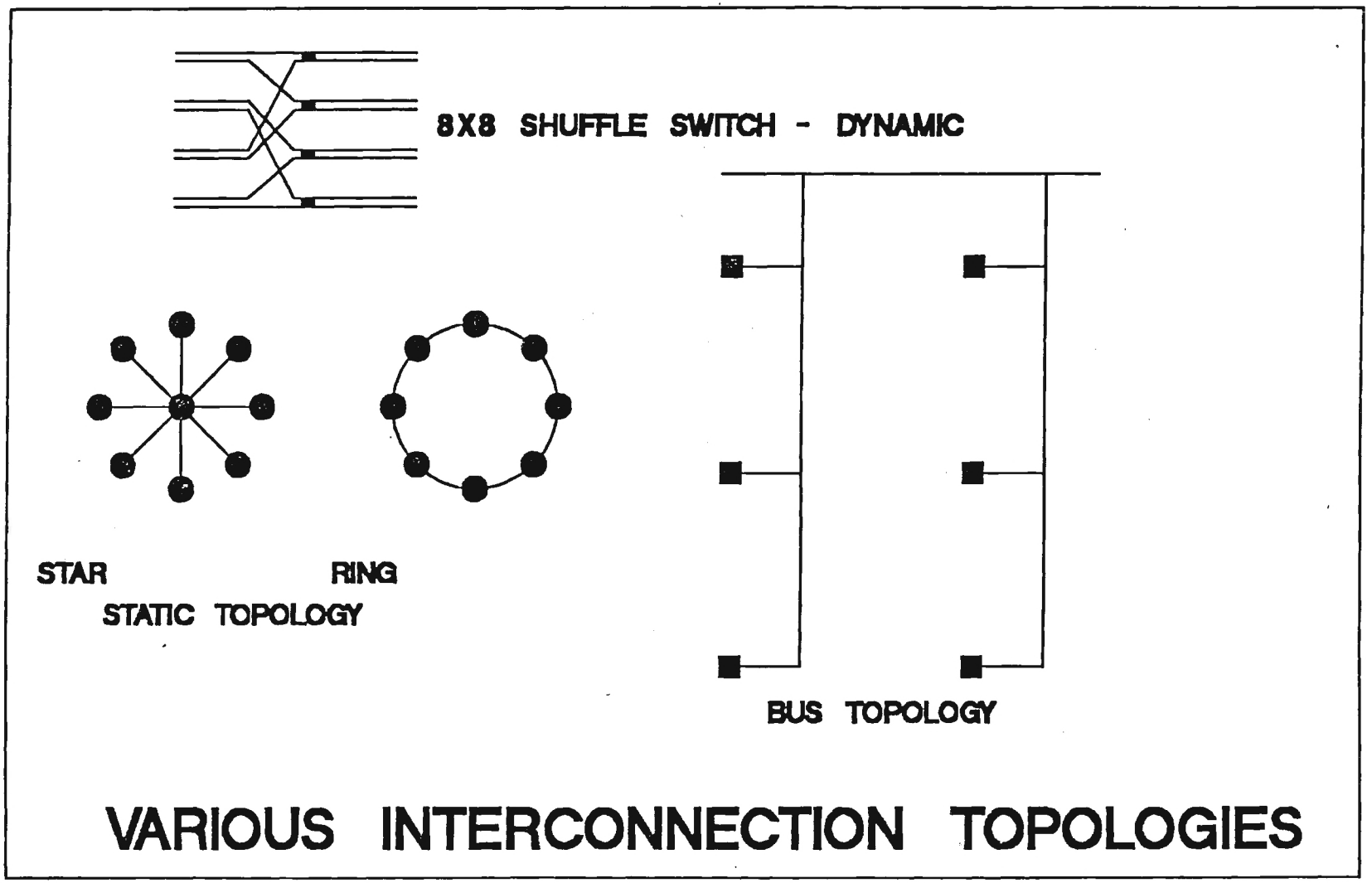
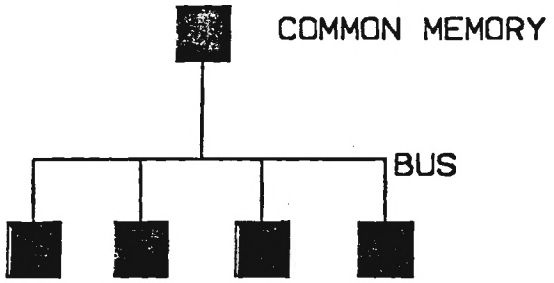
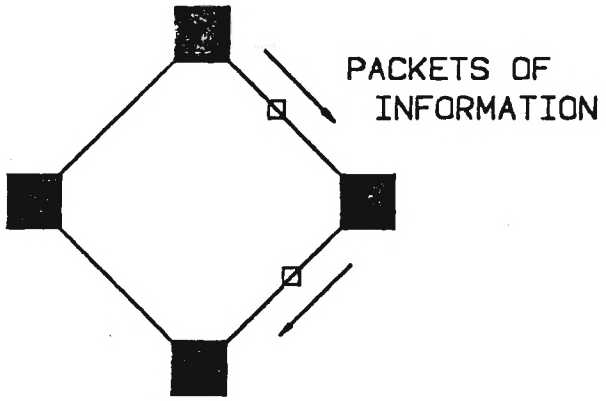


Figure 2. Various Interconnection Topologies.





- READ AND WRITE OPERATIONS IN CM
- RESOURCE CONTENTION
- HW/SW LOCKS NEEDED FOR DATA INTEGRITY
- GOOD FOR ENGINEERING DATA STRUCTURES

- BY MESSAGE PASSING
- NO RESOURCE CONTENTION
- NO LOCKS ARE NEEDED
- UNSUITABLE FOR ENGG DATA STRUCTURES

INFORMATION SHARING MECHANISMS

Figure 3. Information Sharing Mechanisms.

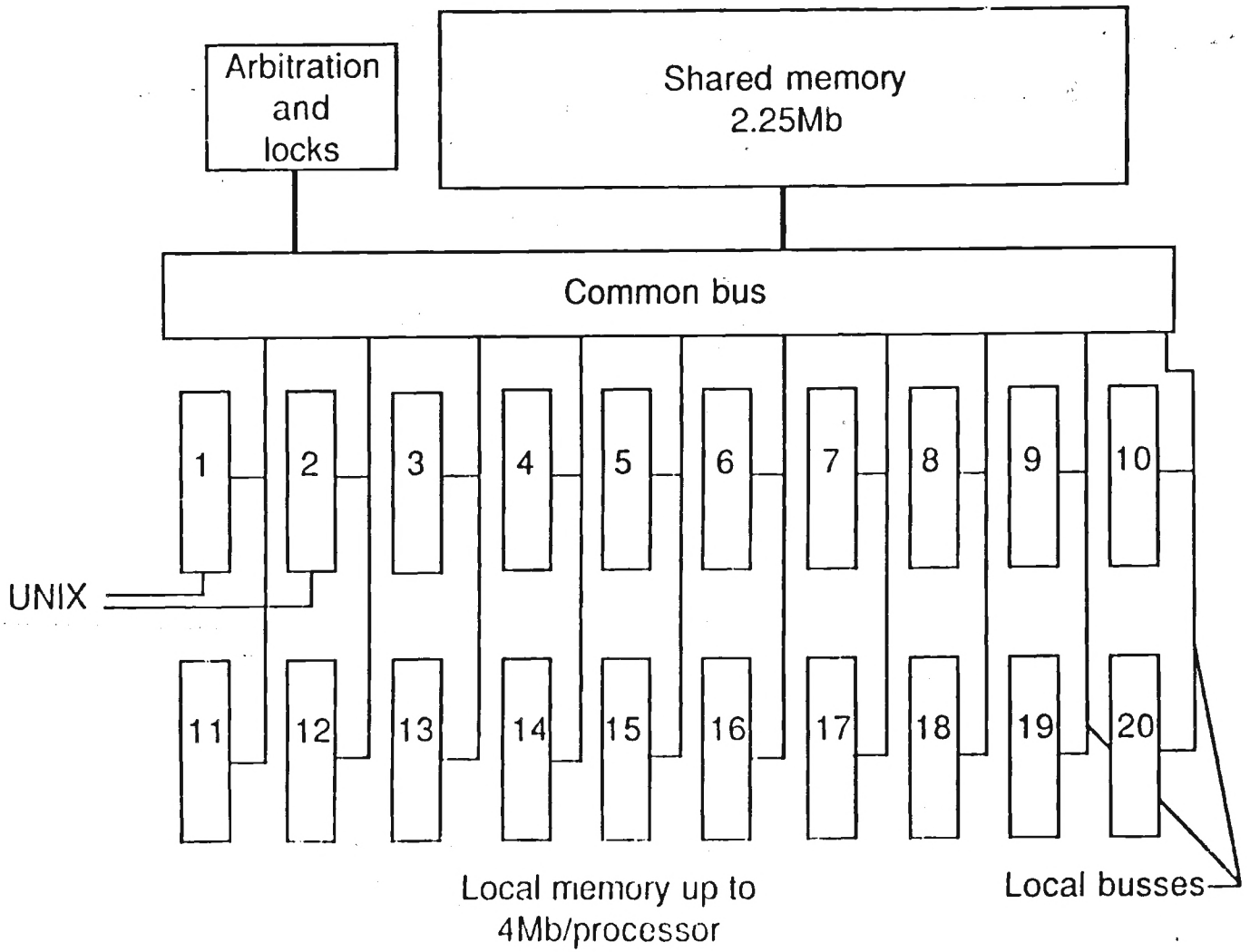


Figure 4. FLEX/32 20-Processor Configuration.

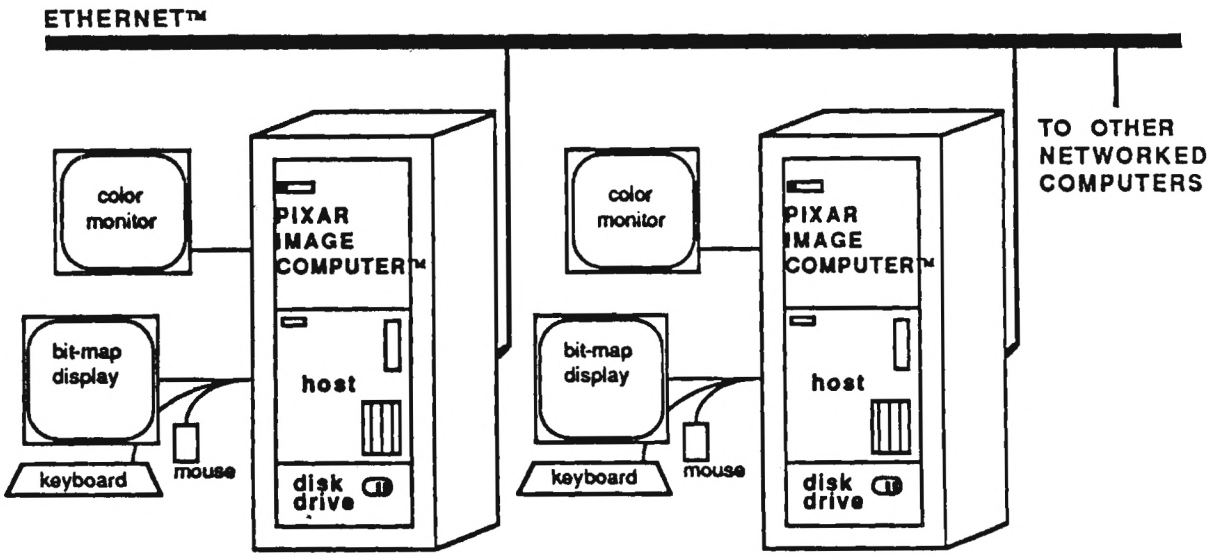


Figure 5a. Pixar Computer and Hosts.

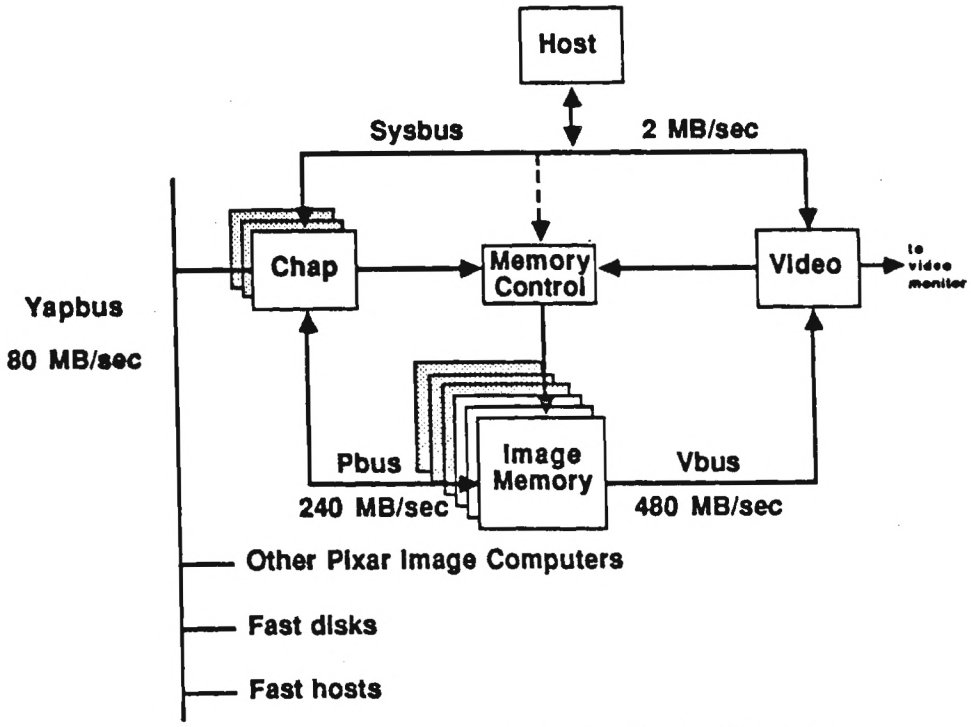


Figure 5b. System Block Diagram

Figure 5. Pixar Image Computer.

RESEARCHERS PARALLEL COMPUTER REQUIREMENTS

* HIGH LEVEL LANGUAGE INTERFACE
* DYNAMIC TASK INITIATION (DETERMINED BY DATA)
* DYNAMIC CREATION OF COMMUNICATION PATHS BETWEEN TASKS
* SEVERAL GRANULARITIES OF CONCURRENCY
* LARGE CONVENIENT STORAGE FACILITIES
* COMPUTATION
    * ABILITY TO HANDLE A NUMBER OF COMPUTATIONALLY INTENSIVE TASKS
    * SOME VECTOR PROCESSING CAPABILITY
* COMMUNICATION
    * LARGE IRREGULAR MESSAGE BURST (> 1KBYTE)

Figure 6. Researchers Parallel Computer Requirements.

ENGINEERS PARALLEL COMPUTER REQUIREMENTS

* ACCESS TO LARGE DATA BASE
* ACCESS TO GEOMETRY SYSTEM
* ACCESS TO A VARIETY OF ONLINE ANALYSIS METHODS
* CONVENIENT, FRIENDLY USER INTERFACE
    * BUILD/ANALYZE/STORE/RETRIEVE ETC
* SYSTEM RELIABILITY: LARGE PROBLEM, ACCURATE RESULTS
* LARGE DEGREE OF CONCURRENT CALCULATIONS

Figure 7. Engineers Parallel Computer Requirements.

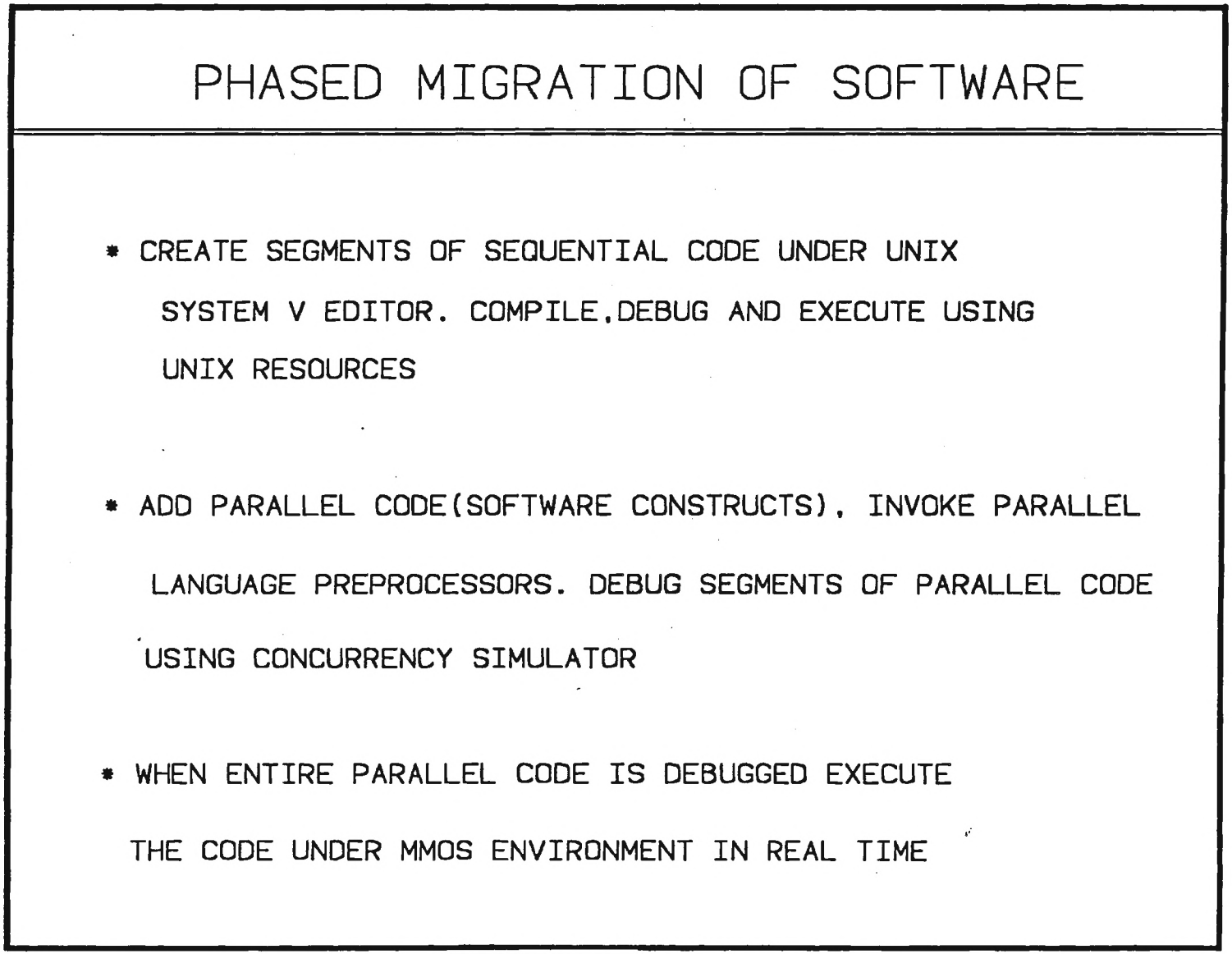
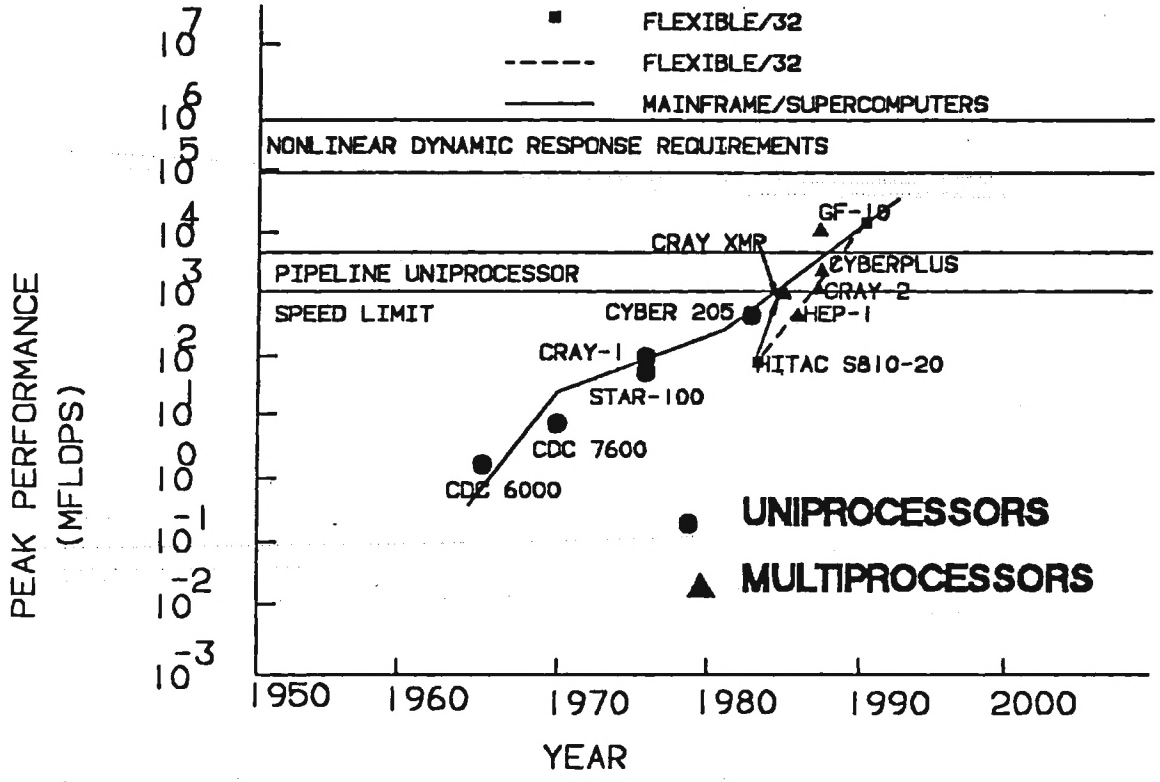


Figure 8. Phased Migration of Software.



GROWTH IN COMPUTER SPEED

Figure 9. Growth in Computer Speed.

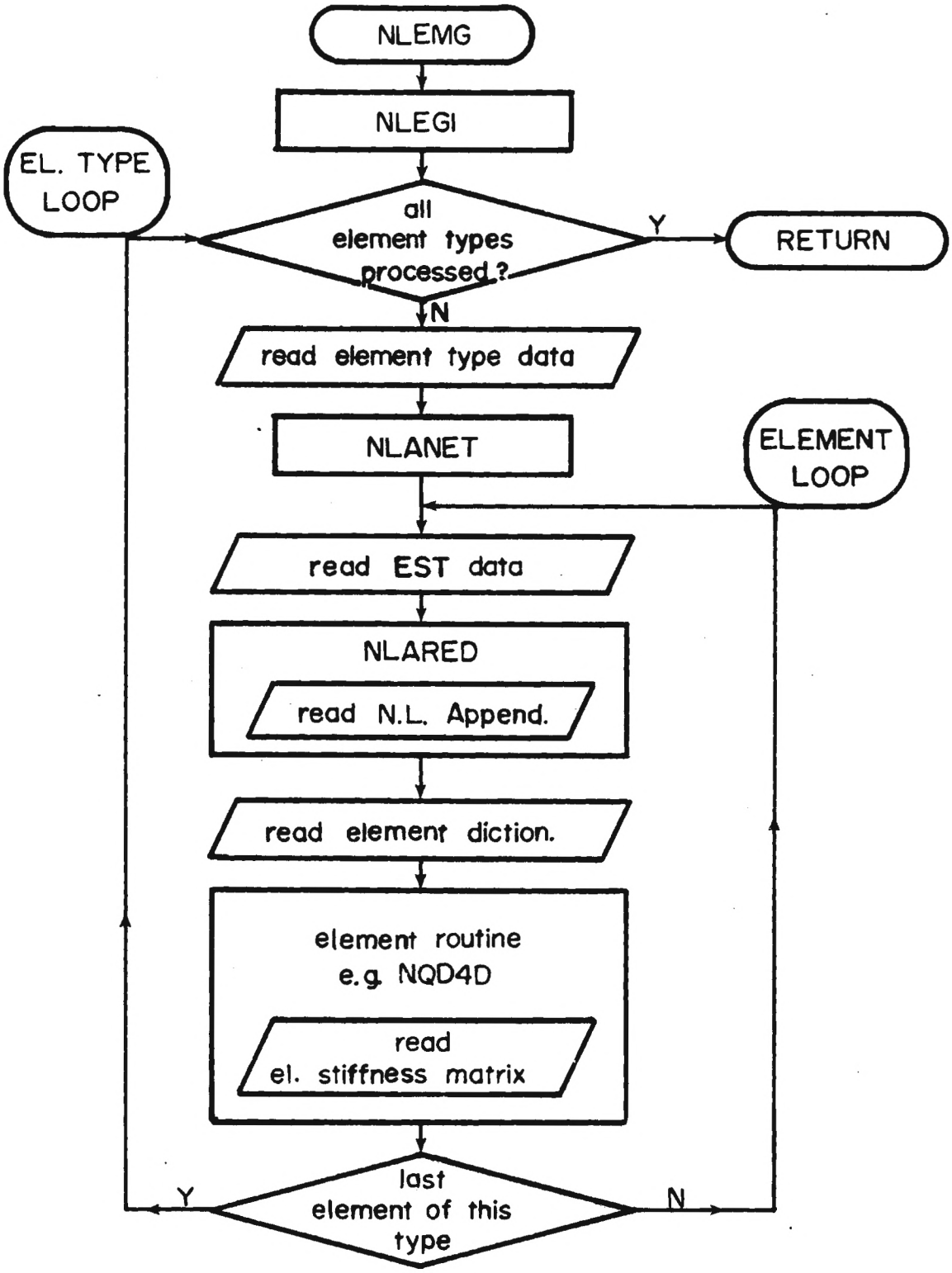


Figure 10. Simplified Flowchart of the NLEMG Module, Dependent Routines and File Access.

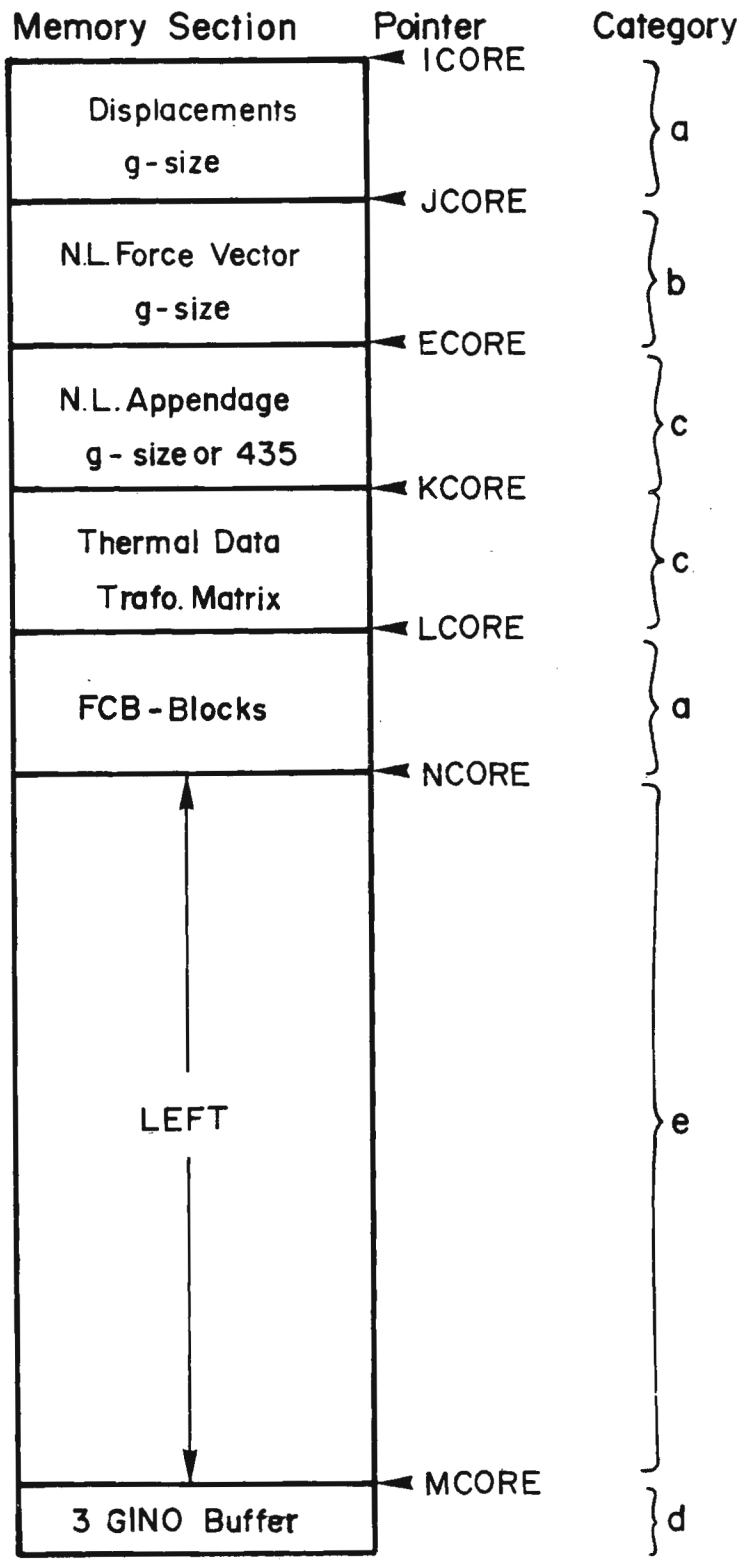


Figure 11. Open Core Memory Layout in NLEMG, Pointers and Categories of Use.

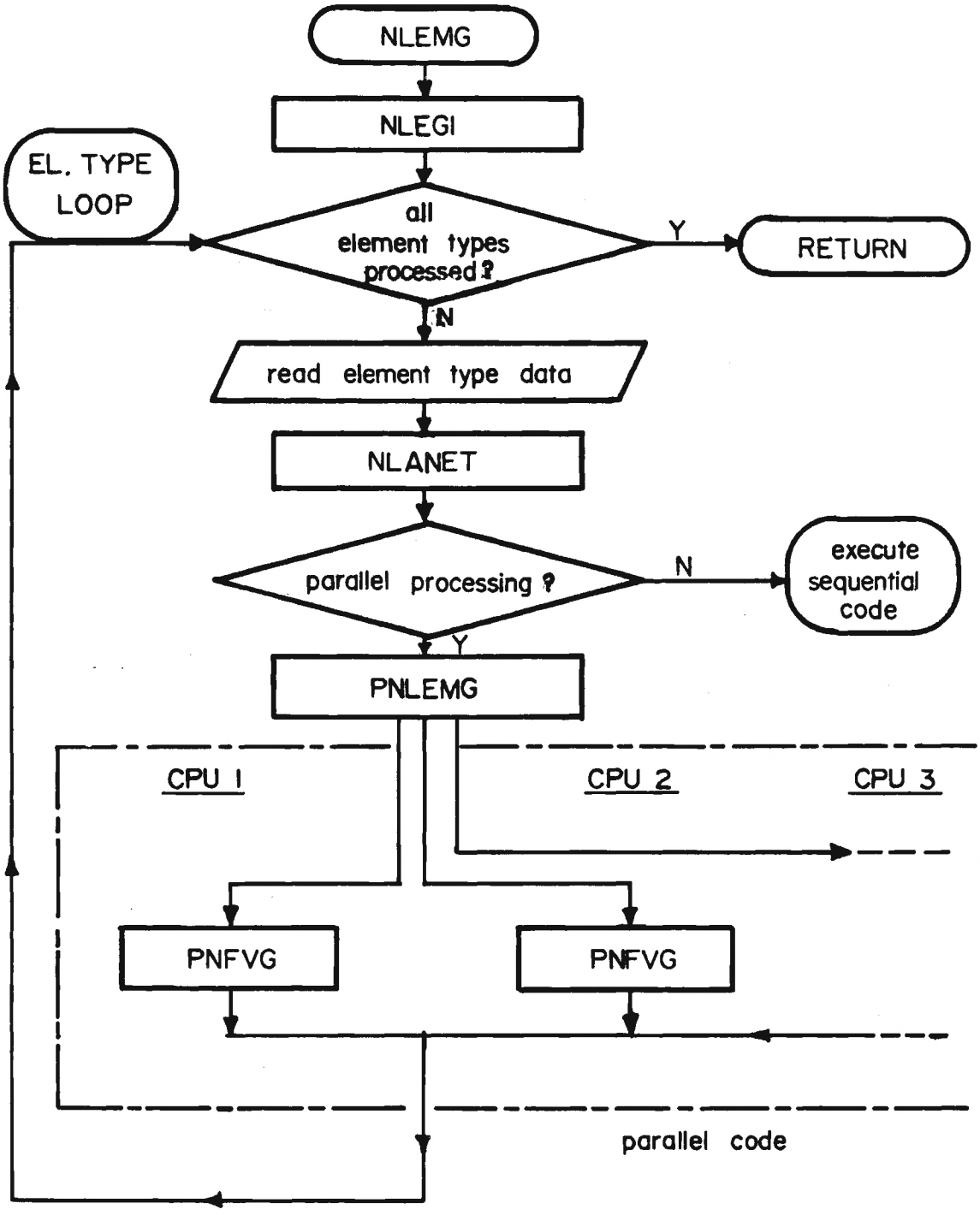


Figure 12. Parallel Processing within the NLEMG Module.

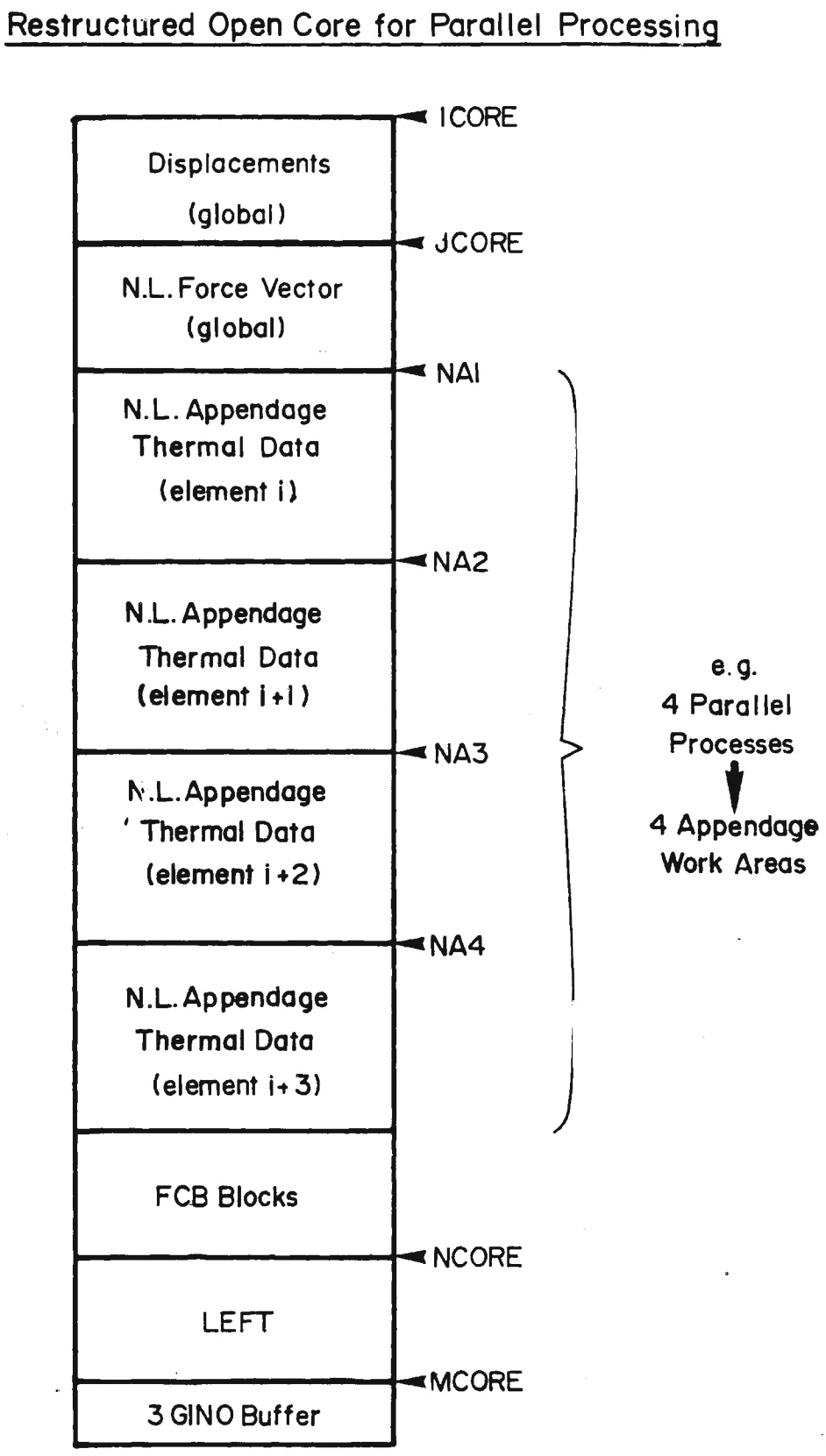


Figure 13. Restructured Open Core Memory for Parallel Processing.

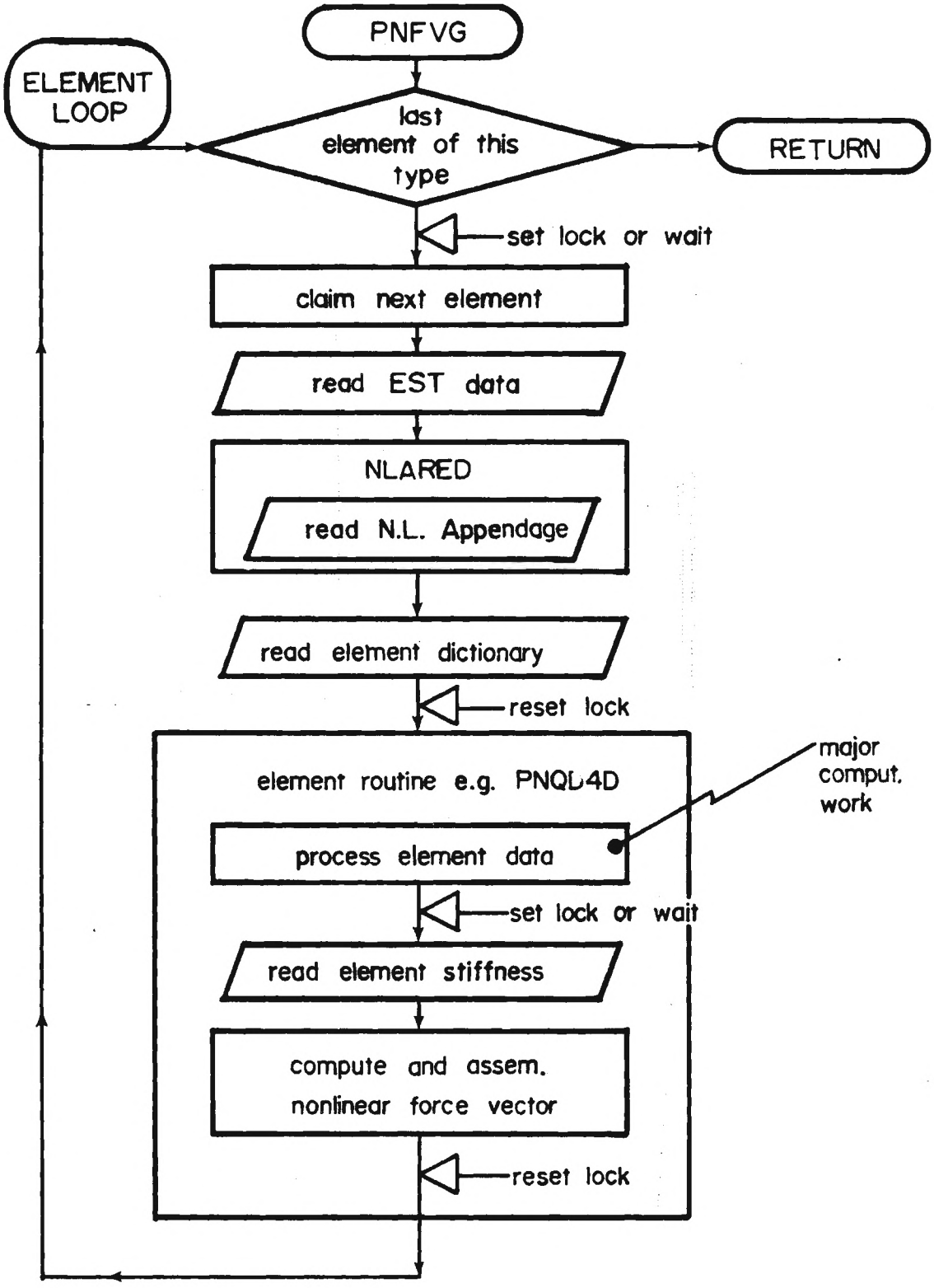


Figure 14. Parallel Nonlinear Force Vector Generation, PNFVG Routines Simplified Flowchart.

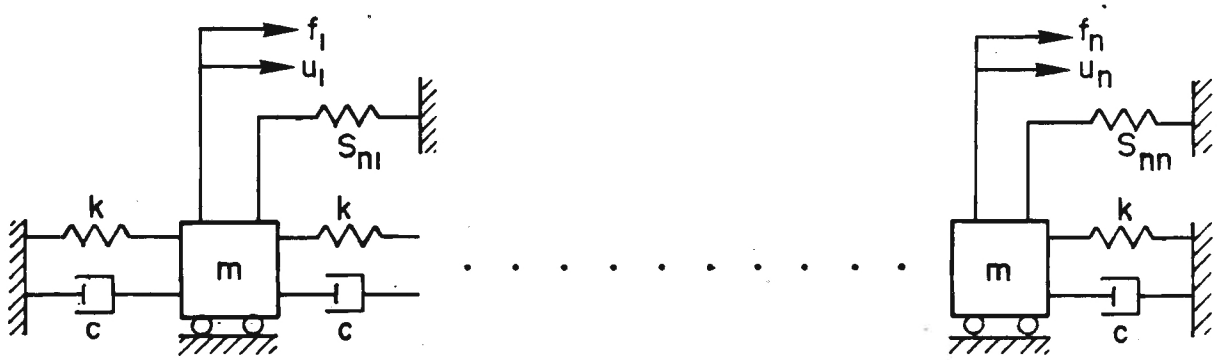


Figure 15. A Sample Problem.

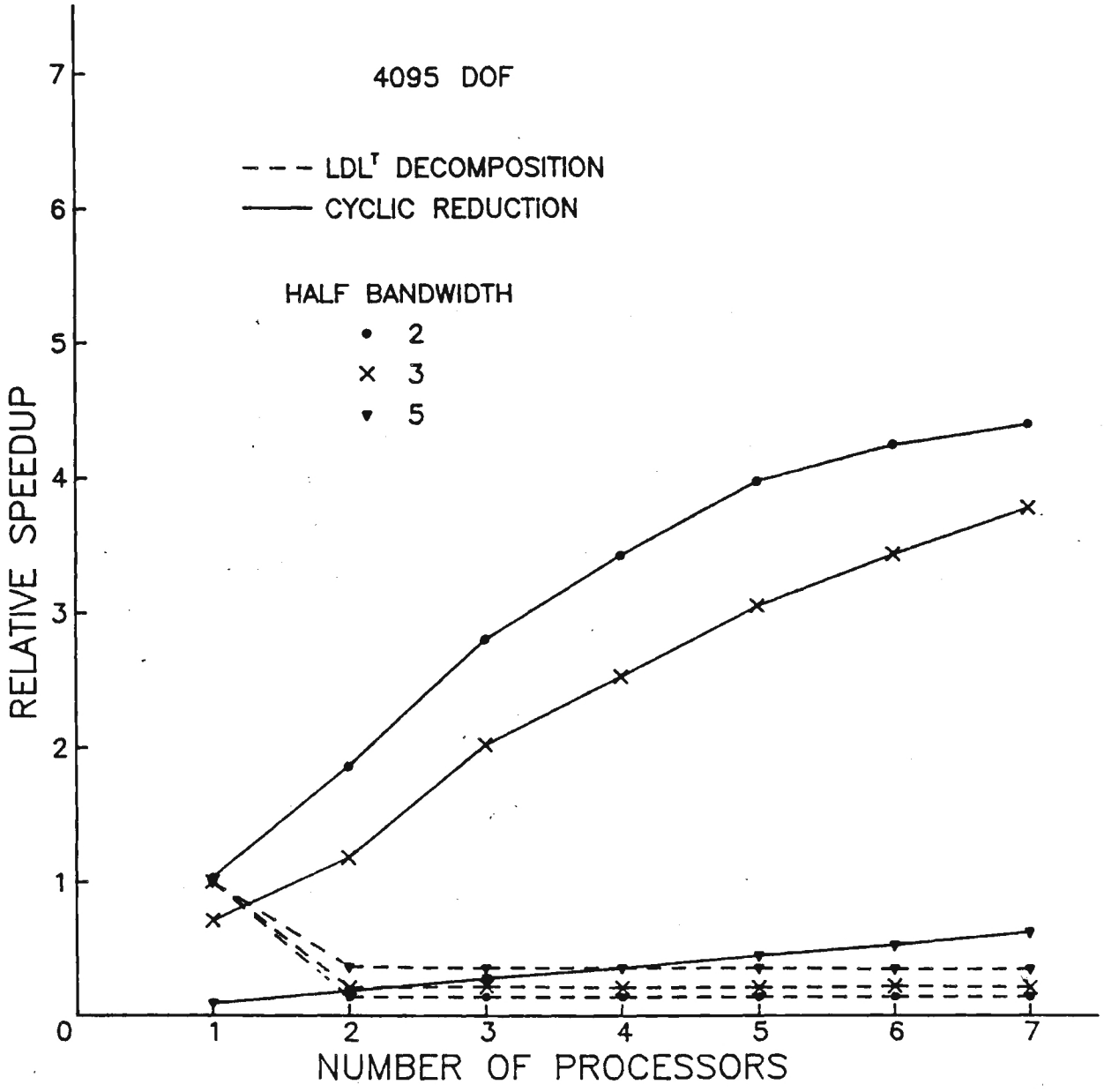


Figure 16. Cyclic Reduction Versus LDL^T Decomposition.

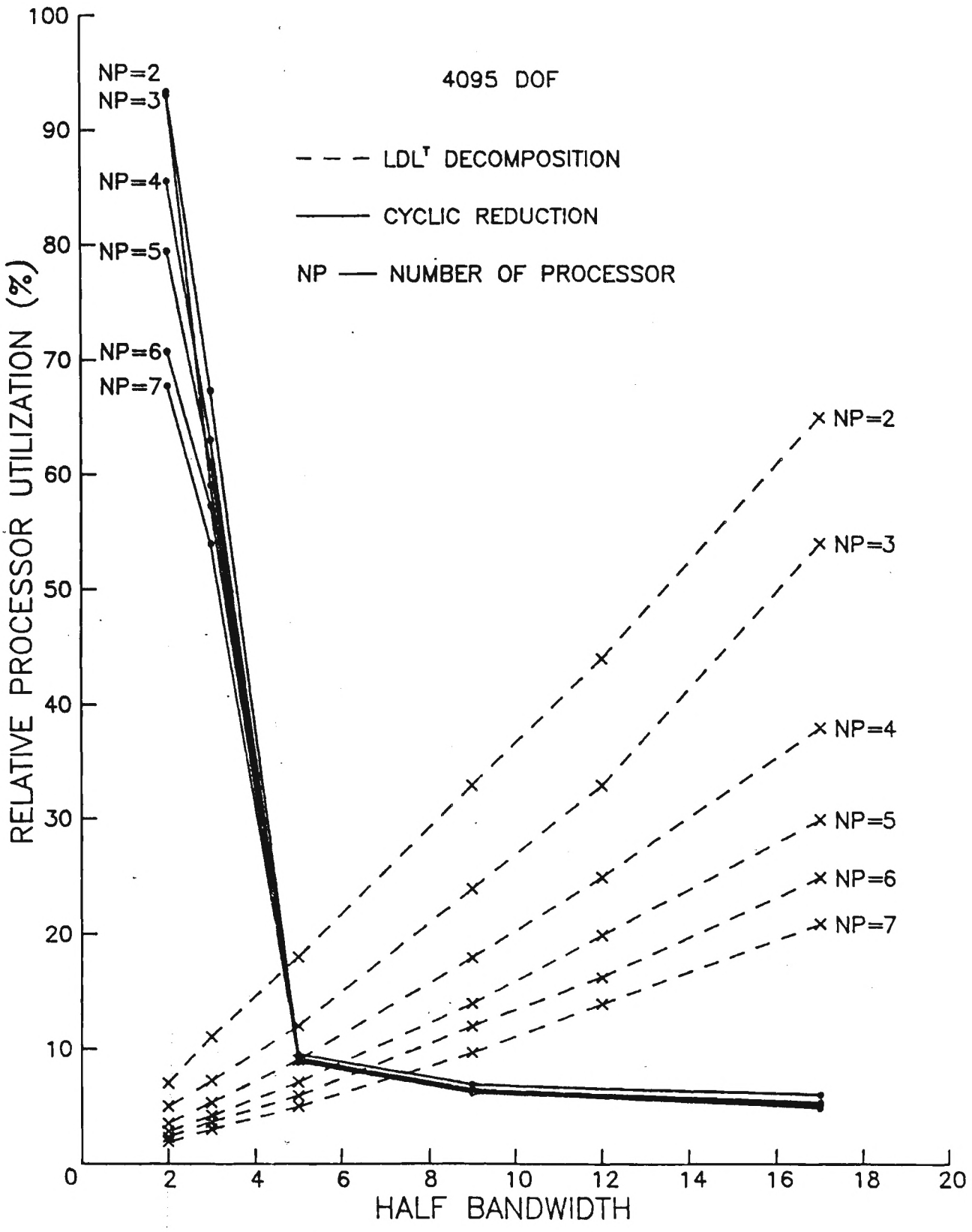


Figure 17. Cyclic Reducton Versus LDL^T Decomposition.

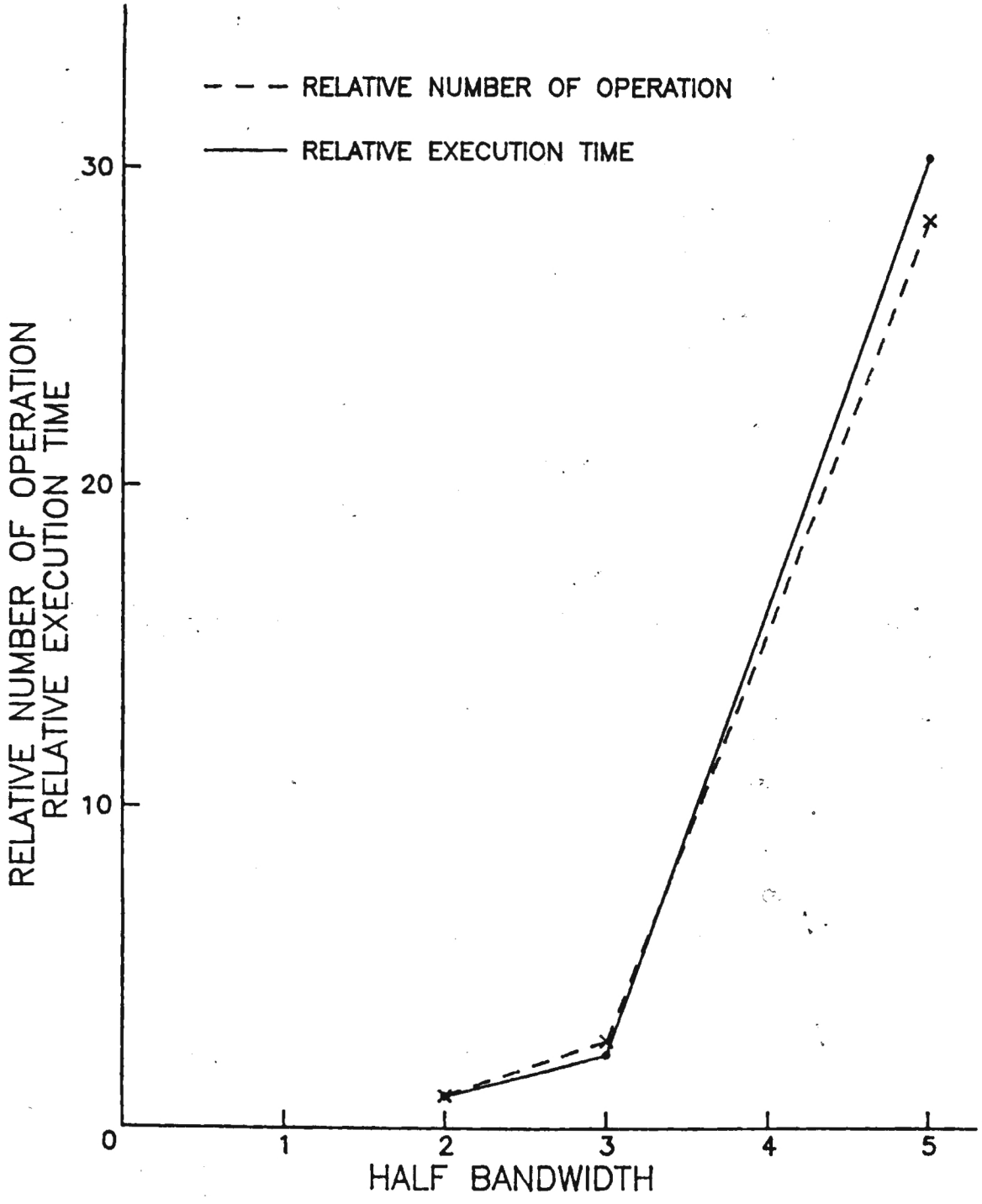


Figure 18. Relative Number of Operation and Relative Execution Time for Cyclic Reduction.

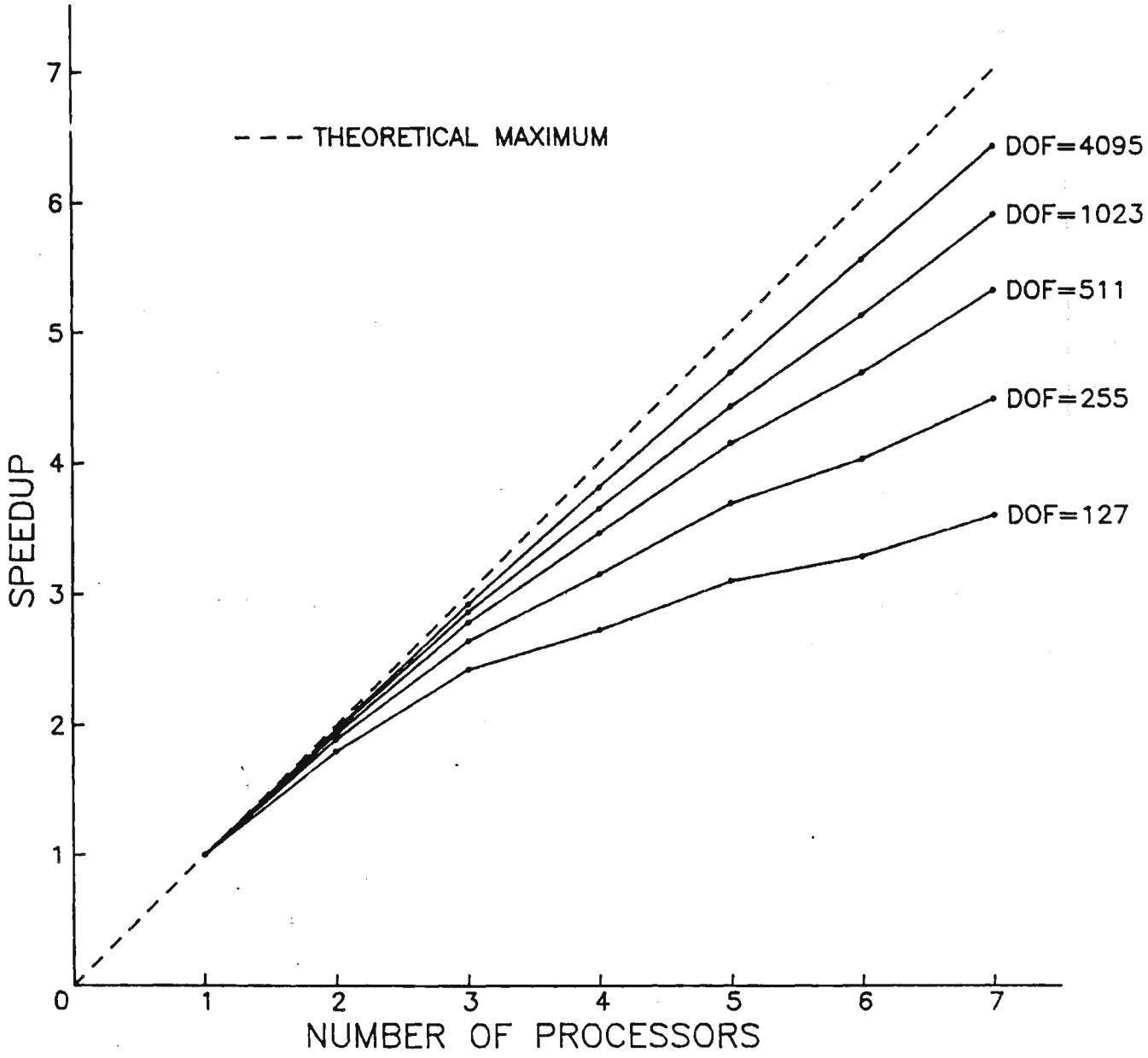


Figure 19. The Newmark-β Method Incorporating Cyclic Reduction.

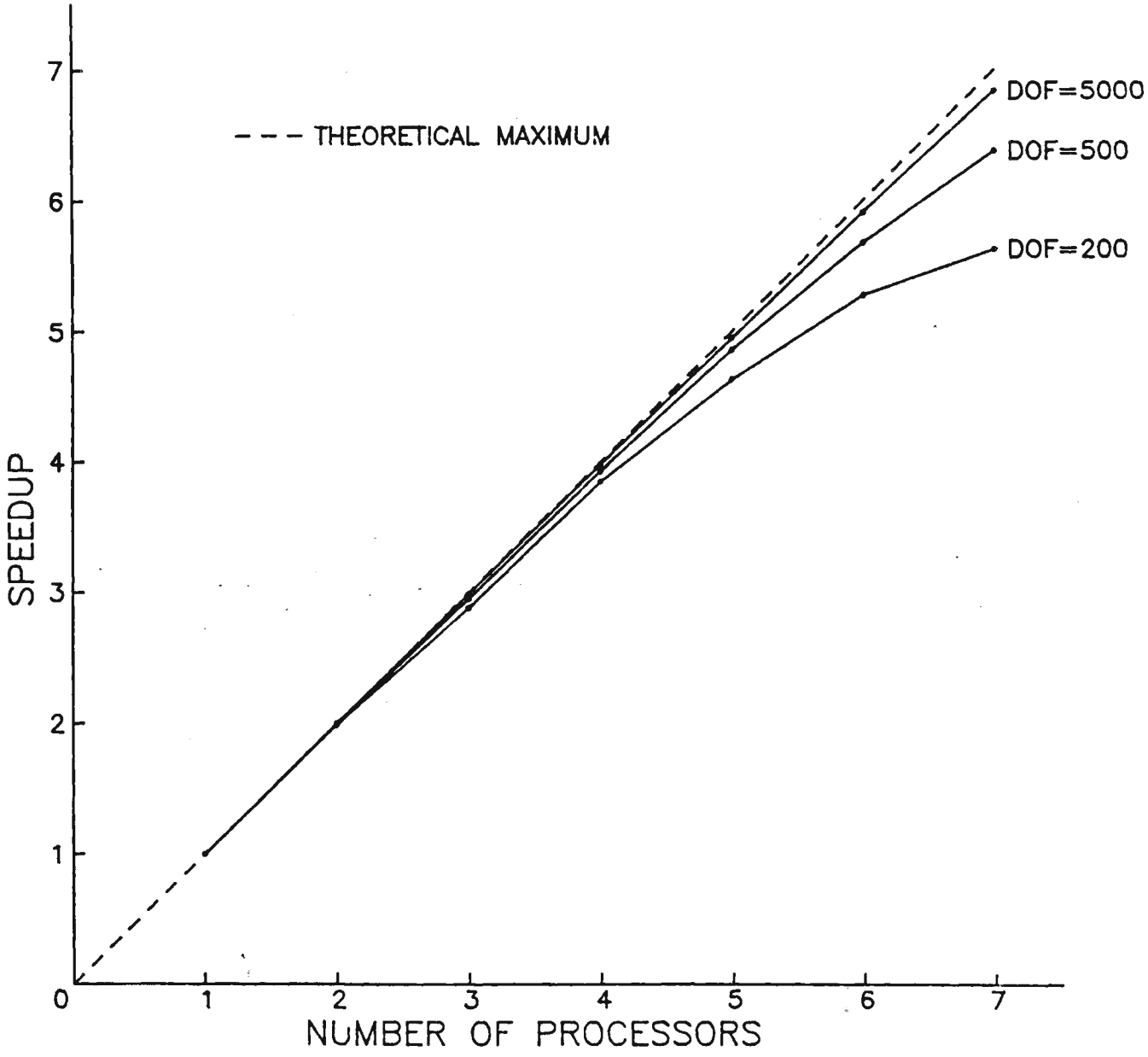


Figure 20. The Central Difference Method: In the Case of Diagonal Mass Matrix.

Solve $Ax = b$

Processor #1 (Gauss-Seidel):

$$x_1 = \frac{1}{a_{11}} (- a_{12}x_2 - a_{13}x_3 - a_{14}x_4 + b_1)$$

$$x_2 = \frac{1}{a_{22}} (- a_{21}x_1 - a_{23}x_3 - a_{24}x_4 + b_2)$$

Processor #2 (Gauss-Seidel):

$$x_3 = \frac{1}{a_{33}} (- a_{31}x_1 - a_{32}x_2 - a_{34}x_4 + b_3)$$

$$x_4 = \frac{1}{a_{44}} (- a_{41}x_1 - a_{42}x_2 - a_{43}x_3 + b_4)$$

Jacobi

Figure 21. Mixed Jacobi/Gauss-Seidel Method.

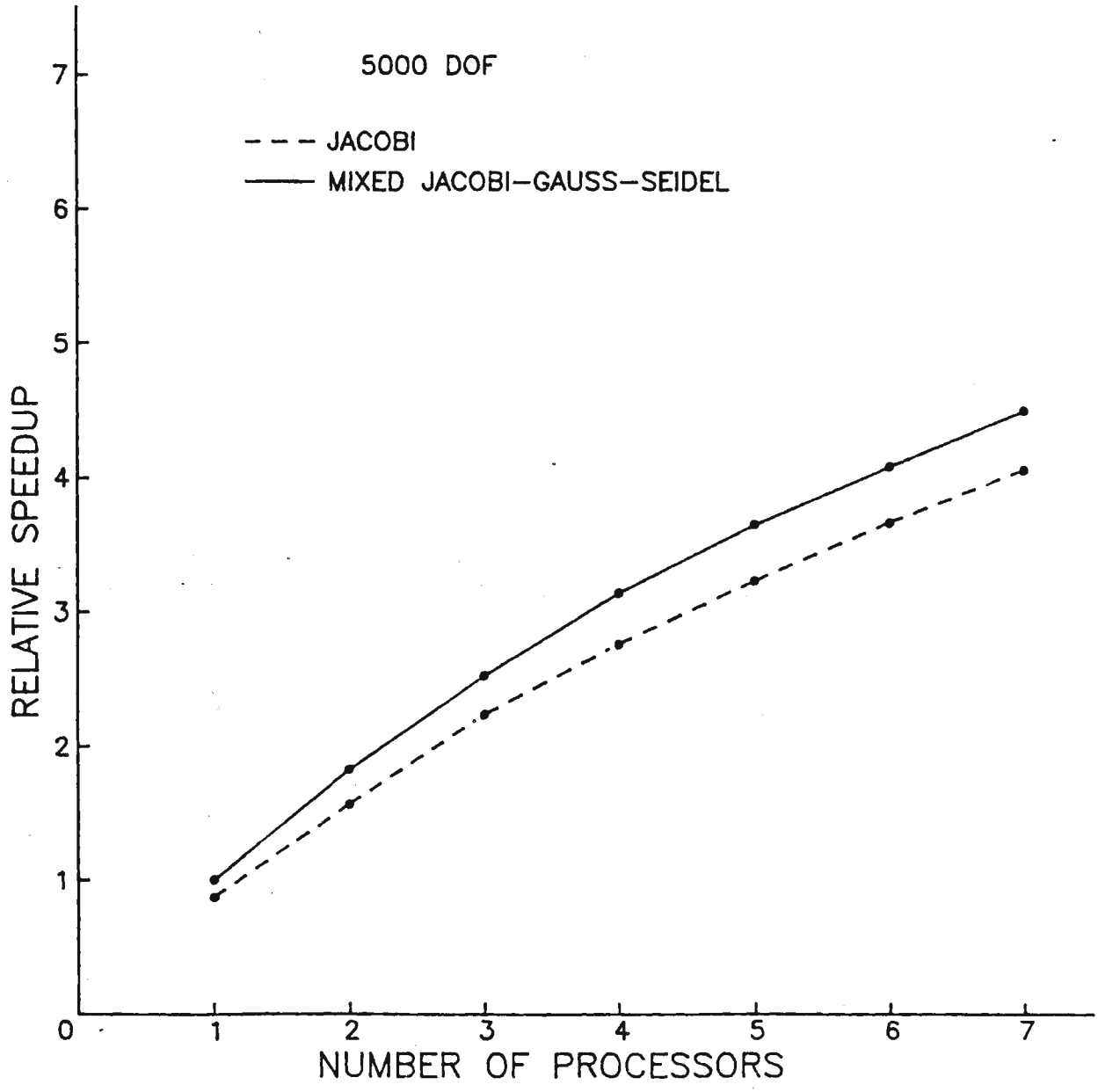


Figure 22. Central Difference Method: In the Case of Nondiagonal Mass Matrix (Iteration Scheme).

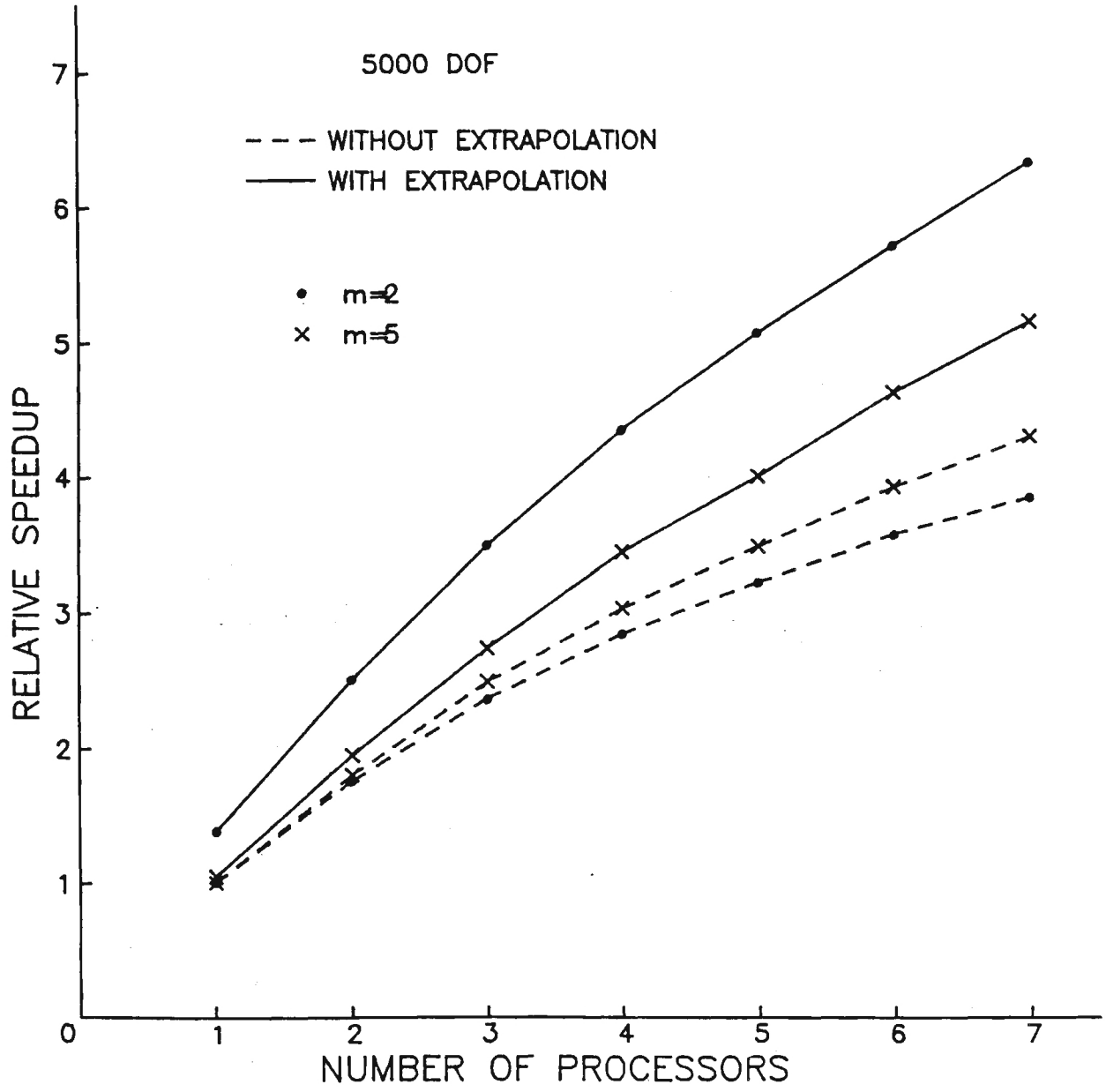


Figure 23. The Central Difference Method Incorporating the Mixed Jacobi/Gauss-Seidel Method (Nondiagonal Mass Matrix).

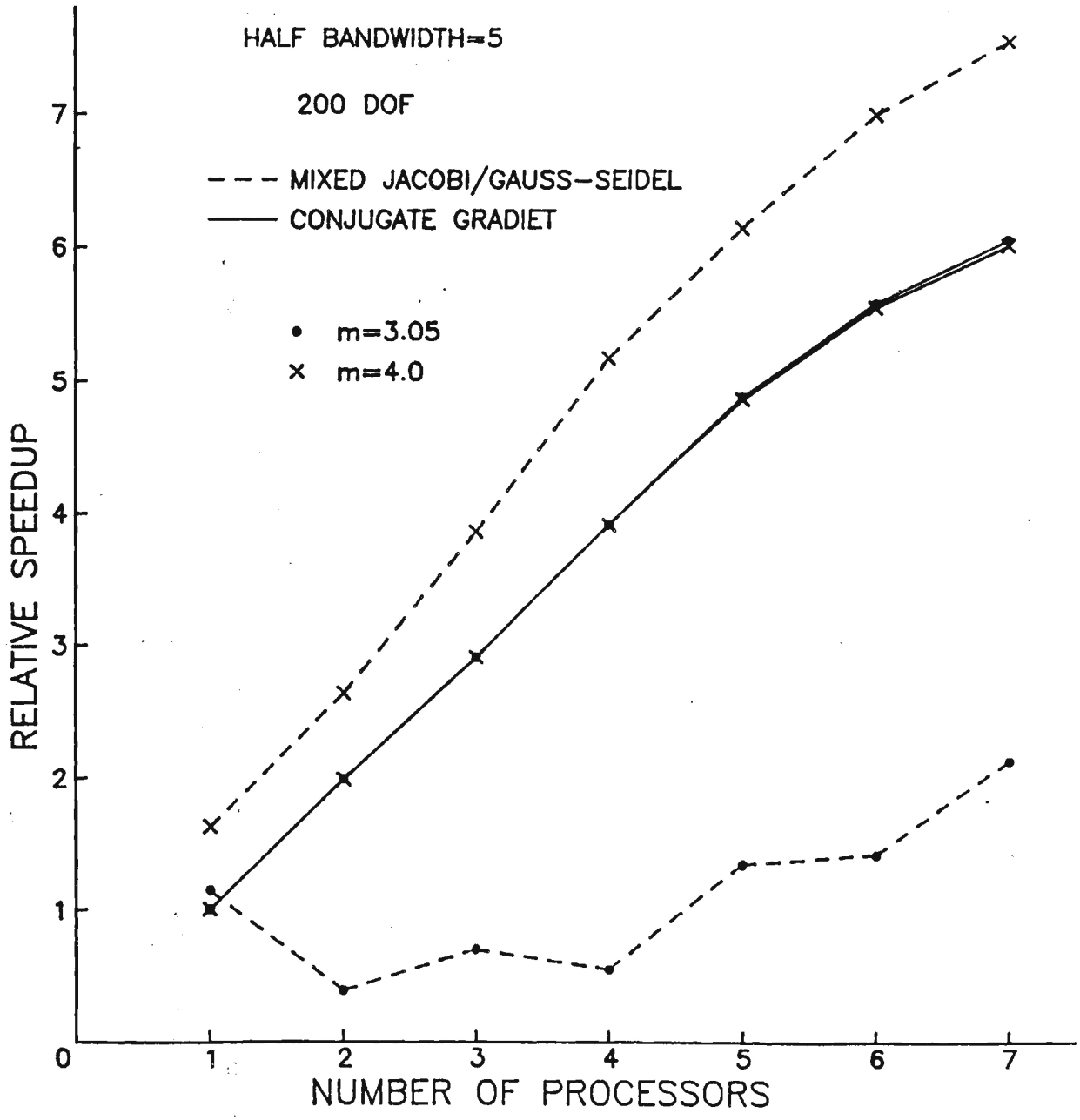


Figure 24. The Jacobi/Gauss-Seidel Method Versus The Conjugate Gradient Method.

E-25-M21

Final Report
Phase II

STRUCTURAL DYNAMICS METHOD FOR PARALLEL SUPERCOMPUTERS

By

Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

Submitted to

The MacNeal-Schwendler Corporation
Los Angeles, California

Under:
Georgia Tech Research Corporation E25-M21
to MacNeal Schwendler Corporation

October, 1988

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF MECHANICAL ENGINEERING
ATLANTA, GEORGIA 30332

STRUCTURAL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

by

Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

Final Report Phase II
Submitted to

The MacNeal-Schwendler Corporation
Los Angeles, California

Georgia Tech Research Corporation E25-M21
to MacNeal-Schwendler Corporation

October, 1988

STRUCTURAL DYNAMICS METHODS FOR PARALLEL SUPERCOMPUTERS

by

Robert E. Fulton
Dietmar Goehlich
Rongfu Ou

October 1988

TABLE OF CONTENTS

Section

SECTION I

INTRODUCTION

The structural design/analysis of future advanced systems in such fields as transportation, energy and aerospace require increasing levels of detail and design complexities to meet mission requirements. Such required analysis may include the combined effects of dynamic loads, composite materials, multidisciplinary interactions, three dimensional geometry and nonlinear behavior. Realistic structural models for such designs typically imply large-order finite element methods and excessively large computational requirements. Since the advent of NASTRAN, general-purpose finite element structural analysis computer programs have provided the capability to address a wide range of structures problems. However, when these programs are applied to nonlinear problems, the limitations of sequential computers result in reduced scale models or excessive computing times. For example, the routine dynamic analysis of large scale structural problems is not feasible on current sequential computers, and many future engineering designs will require effective speeds greater than 10^3 MFLOPS (million floating point operations per second).

Projected advances in computer technology indicate signficiant increases in effective calculation speed will be available in the 1990's, through "super computer" architectures consisting of arrays of vector processors operating in parallel on different tasks. Such advanced architectures denoted MIMD (Multiple Instruction, Multiple Data) computers will be available on a wide range of computer hardware from work stations to large mainframes and will have the potential for significantly increasing effective calculation speeds. But this increase in speed cannot be effectively utilized for nonlinear finite element problems without the development and implementation of appropriate numerical algorithms which take advantage of the parallel/vector computation features of this new generation of computers. This research effort focuses on the development and evaluation of parallel/vector methods for dynamic analyses of complex nonlinear finite element structural problems.

The research summarized herein specifically reports on an investigation and evaluation of selected algorithms for MSC/NASTRAN which appear best suited for nonlinear dynamic analyses on parallel supercomputers. It builds on the results of the Phase I study reported in Ref. 1.1. The principal areas of investigation in this Phase II effort are:

1. Investigation of appropriate program designs and data structures for vector and parallel processing.
2. Investigation of Data Management requirements for MSC/NASTRAN to support parallel finite element computations.
3. Investigation of decoupled parallel numerical integration approaches for nonlinear dynamics.

The following sections address each of these ares. Sections II and III cover the first area; Section IV addresses the second area; and Section V addresses the third. Specific summaries or conclusions relative to each of these areas is included at the end of the respective sections; the overall study conclusions and recommendatons are given in Section VI.

1.1 REFERENCES

1.1 Fulton, R. E., Goehlich, D., Ou, R., "Structural Dynamics Methods for Parallel Supercomputers", Final Report - Phase 1, submitted to The MacNeal-Schwendler Corporation, October, 1987.

SECTION II

RELEVENT PROGRESS IN PARALLEL HARDWARE

Significant advances have taken place over the past year in parallel/vector supercomputer hardware which can provide directions relative to finite element computations and appropriate software. It is useful to classify the evolving supercomputer family into four categories, moderately parallel high speed systems, massively parallel systems, moderately parallel intermediate systems, and high performance parallel work stations. Figure 2.1 gives a list of the parallel/vector computers in the four classes.

Figure 2.1 shows that at one extreme are the high speed moderately parallel systems such as the CRAY and IBM systems. At the other extreme are the massively parallel systems such as the hypercube or Butterfly systems. In between are various intermediate systems such as the Alliant and Convex systems. The fourth group corresponds to the high performance parallel/vector workstations, a number of which were introduced at SIGGRAPH 88 (August/88). This latter group indicates the evolution of the established workstation market toward the new technology of parallel/vector processing; Figure 2.1 gives data on speed and memory for these systems.

During the past year several events have occured which give an indication of the opportunities for FEM software on commerical parallel/vector computers. For example, supercomputer conferences were held in the U.S., Norway and Japan (Figure 2.2). Especially of interest is the noticeable change in IBM's posture relative to parallel computers. Until recently IBM largely ignored parallel processing but IBM is now marketing the IBM 3090 as a parallel computer with 6 processors. Steve Chen, formerly of CRAY, now leads an IBM design team rumored to produce competition for CRAY. IBM's Bergen Scientific Center sponsored the Norway conference focused largely on parallel technology. These and other IBM actions make it clear that IBM now considers parallel processing to be a significant market and their action makes parallel computing credible. The IBM action

will quickly influence the Japanese supercomputing market which to date has concentrated only on very high performance vector processors.

Finally CRAY has announced its plans for the future (Figure 2.3) which indicate a growing level of parallelism to go with its vector and large memory capability. The Pittsburg Supercomputer Center has already committed to obtaining the first 16 processor CRAY-3 to be delivered in 1989 and CRAY projects a 64 processor CRAY-4 in 1992.

These results lead to several signficiant conclusion relative to the future of parallel processing.

1. The moderately parallel large scale computers are an established applications market with CRAY now selling 50% of its computers to industrial organizations. Furthermore, CRAY is moving steadily toward increasing numbers of parallel processors. The ETA and IBM activities make it a competitive market. To date the Japanese (Hitachi, Fujitsu and NEC) have focused on maximizing vector capabilities but the IBM strategies and the Tokyo Conference indicate that Japan will soon expand to parallel architectures.
2. The massively parallel computer market (e.g. hypercube, Butterfly, etc.) is not yet well established and usages are basically research oriented (some of it FEM oriented). Interest is still growing but the absence of general purpose FEM software for such machines inhibits their practical use. Nevertheless there is a growing set of special purpose software; some eye catching FEM results have begun to evolve (e.g. Sandia, Fig. 2.2); and the vendors are marketing aggresively.
3. The moderately parallel intermediate systems market (e.g. Alliant, Convex) is still evolving and these systems are being used as less expensive alternatives to the CRAY or as CRAY frontends. Informal projections are that Alliant will soon move to a 16 or 32 processor capability.
4. The recent explosion on the scene (Fig. 2.1) of a large number of moderately parallel/vector based high performance workstations is an especially important event and is an indication of the future. Both the Ardent and Raster systems have parallel and vector capability and Apollo and Raster have rated

performance exceeding 100 MFLOPS. Such systems provide a parallel/vector graphics and computation capability on the engineering desk. These trends suggest that high performance parallel workstations will soon be highly competitive with the parallel intermediate systems such as the Alliant and Convex. This would be similar to that which occured when workstations began to take over much of the minicomputer market for sequental processors.

These results indicate the importance of MSC to continue to position itself with respect to parallel computing. MSC FEM software is needed which is tailored to large supercomputers such as CRAY with both parallel and vector capability since these computers will continue to increase in parallel processors. The software can also be installed on the moderately parallel systems such in the Alliant. More important the trends suggest that the parallel MSC FEM software needs to be downscaled to run on the the moderately parallel high performance workstations, where extra processors may be available to the user at no cost penalty. These hardware advances suggest it likely that in the next five years most computers hosting MSC/NASTRAN will contain both vector and parallel capability.

The results also indicate the continued evolution of the massively parallel computer market. This may occur though the acceptance of innovative minicomputer systems or by attached accelerators based on the hypercube or Butterfly configurations. The massively parallel approach for the high speed systems also appears to be steadily occuring through the continued growth of the CRAY type general purpose computers, with CRAY planning at least 64 processors in 3-4 years. IBM is also known to be studying massively parallel architectures. These results strongly indicated that the hardware base will evolve for a large number of parallel/vector processors. Commercial finite element systems in 3-5 years will be needed which run effectively on 100-1000 processors and MSC needs to position itself relative to that market through existing or future software.

Figure 2.1
Parallel Computer Family

	Processors	Vector	*Memory
High Speed Systems			
Cray Y-MP	8	x	S
IBM 3090	6	x	S
ETA	8	x	S/L
Massively Parallel			
BBN Butterfly	256		L
INTEL iPSC/2	128	x	L
N Cube	1024		L
Amatek	1024	x	L
Connection	65,000		L
Intermediate Systems			
Alliant FX/80	8/16	x	S
Sequent Balance	12	x	S
Convex	4	x	S
FLEX/32	20/40	x	S/L
High Performance Workstations			
Ardent TITAN	4 (64)**	x	S (128)***
Stellar GS-10000	4 (40)	-	S (128)
Raster/Sun	8 (160)	x	L (128)
Apollo DN-10000	4 (144)	-	S (128)
Pixel	82 (820)	-	S (128)
Silicon Graphics	2 (40)	-	L (41)

*S = Shared
L = Local

** MFLOP Speed

*** MBYTE Memory

Figure 2.2
Other Parallel Computer Activities

IBM Becoming More Visible
- Chen Joins IBM to Lead New Initiative
- 3090 Being Marketed as Parallel
- IBM Parallel Fortran
- Bergen Scientific Center, Norway, Conf.
- Making Parallel Computing Credible

NASA Langley Developing a CRAY Type FE System
- Documented, Prototype Software
- Buying Convex/CRAY 2 system

Several Parallel Conferences
- Boston — Computer Science Focus
- Tromso, Norway — Lots of Hypercube Activities
- Tokyo — Graphics Workstation

Vendors Marketing MSC Parallel LDL^T Decomposition
- IBM, Alliant, Convex

Sandia Produces Massively Parallel Structures FEM Example of Plane Stress Cantilever Beam
- Demos 1024 Processor NCUBE Hypercube which Gives 500 Speedup Over 1 Processor Solution

Figure 2.3
CRAY Supercomputer History/Plans

Machine	Date	Number of Processors	Rated Speed MFLOPS	Memory MWORDS
CRAY-1	1976	1	160	1
X-MP/2	1982	2	420	4
X-MP/4	1984	4	840	8
CRAY-2	1985	4	1,700	128
Y-MP	1988	8	2,500	32
CRAY-3	1989	16	16,000	572
CRAY-4	1992	64	128,000	2048

SECTION III

PROGRAM DESIGNS AND DATA STRUCTURES FOR VECTOR PROCESSING

Todays supercomputers (e.g. CRAY, IBM3090) and also many modern mini-computers (e.g. CONVEX, ALLIANT) are equipped with special vector hardware for high speed data processing. Conventional computers have a single instruction stream operating on a single data stream (SISD) while vector computers have a single instruction stream operating on multiple data (SIMD). The vector/scalar speed ratio can reach values of 10 and more for certain computers. To exploit the superior vector performance of such machines, data have to be structured in computer vectors and operations have to be performed in loops over the computer vectors. The scalar/vector performance ratio is in general a function of the vector length, i.e. with increasing vector length the execution time per term in the vector is reduced. Since the vector registers have a specific length (e.g. 64 words) it is most efficient to process vectors which are a multiple of the register length (e.g. 64, 128 etc.). Since the use of vector hardware implies some startup overhead there is a minimum vector length (commonly around 10 words) for which vector processing yields performance gain.

The process of migrating a particular program from scalar to vector processing is commonly denoted as "vectorization" which can be achieved by calling vectorized kernel functions or by using vectorizer compilers. Although compilers commonly provide helpful tools for vectorizing the code, in general special program designs are necessary to generate highly vectorized (and therefore computationally efficient) software. It should be noted that executing a vectorizable program on a SIMD machine without using the vector processing capabilities is simply a waste of computational resources.

Vectorized matrix operations like matrix decomposition, matrix multiplication etc. have been implemented successfully in MSC/NASTRAN by using vectorized kernel functions (e.g. vectorized dot product). Element dependent procedures in NASTRAN's solution sequences (e.g. the EMG phase), however, do not take advantage of vector processing. However, such procedures may require a significant amount of computing time in large scale

analyses. The research effort described in this section has been aimed at program designs and corresponding data structures for vectorized element computations. Two conceptually different approaches have been identified:

- Single element vectorization.
- Multiple element vectorization

Both concepts were subject to investigation in the present study. Earlier work in the field of vectorized element computations is discussed in a brief literature review in the following paragraph. The NASTRAN TETRA element has been chosen as a testbed for a comparative design study. The basic theory of isoparametric solid elements is reviewed and the generation of element stiffness matrices is described in general and in particular for the TETRA element. Program designs and data structures for single and multiple element vectorization are presented and performance data is given comparing the current NASTRAN TETRA element with the two vectorized approaches. Furthermore the implementation of these concepts in a production FEM system is discussed with respect to the MSC/NASTRAN element family, and finally some areas of future work are recommended.

3.1 PREVIOUS WORK ON VECTORIZING ELEMENT COMPUTATIONS

Vectorized element procedures based on single element vectorization have been proposed and tested as early as 1978 by Noor and co-workers [3.1,3.2]. Due to hardware limitations at that time significant speedup could only be achieved for high order elements resulting in long computer vectors. Recently a modified approach has been reported where vectorization is based upon processing Gauss points and also elements in parallel [3.3]. Only minor speedup was reported, but the performance results were obtained on a CYBER 205 which reaches good vector processing performance only on very long vectors untypical for element procedures.

Some general aspects of using vector computers for FEM analysis are reviewed in [3.4] and a first approach to apply vectorization to nonlinear elements has been reported in [3.5].

Multiple element vectorization has been used extensively in the DYNA and NIKE codes developed at Lawrence Livermore National Laboratory [3.6-3.8] and in the PRONTO code from Sandia National Laboratories [3.10]. No detailed study on the performance improvement by vector processing has been published but it was reported that typically a speedup of about 4 can be achieved. The incorporation of vectorized element computations in an element by element preconditioned conjugate gradient method has been reported by Hughes and co-workers [3.9].

3.2 ISOPARAMETRIC SOLID ELEMENTS FOR LINEAR ELASTICITY

The element stiffness matrix of an isoparametric three dimensional element for elasticity problems can be expressed in the form

$$[K]_{el} = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} [B]^T [D]\ [B]\ |J|\ dr\ ds\ dt \qquad (3.1)$$

where [B] is the strain-displacement matrix, [D] is the stress-strain matrix and $|J|$ is the determinant of the Jacobian for the transformation from the element coordinates x,y,z to the intrinsic coordinates r,s,t which range from -1 to +1. $[K]_{el}$ is a symmetric matrix of order [N x M] where N denotes the number of node points and M the degrees of freedom per node.

Introducing Gaussian numerical quadrature in three dimensions equation (3.1) may be recast as

$$[K]_{el} = \sum_{i,j,k} \alpha_i\ \alpha_j\ \alpha_k\ [B]^T\ [D]\ [B]\ |J| \qquad (3.2)$$

The stress-strain matrix and the Jacobian are functions of the spacial coordinates and have to be computed separately for each "Gauss point". The evaluation of eq. (3.2) requires 8 steps which are summarized in the following. The index n (n = 1,...,N) corresponds to the active node points while p (p=1,...,P) denotes the Gauss points. For a solid element generally three nodal displacement but no rotations are monitored, hence M is taken to be three.

STEP 1: SHAPE FUNCTION DERIVATIVES. There are as many shape functions as node points.

$$F_n(r,s,t) = f(r,r_n)\, f(s,s_n)\, f(t,t_n) \tag{3.3}$$

Three partial derivatives have to be computed at each Gauss point, hence there are 3 x P x N terms to be evaluated.

$$Fr_n = \frac{\partial F_n(r_i,s_j,t_k)}{\partial r} \quad ; \quad Fs_n = \frac{\partial F_n(r_i,s_j,t_k)}{\partial s} \quad ; \quad Ft_n = \frac{\partial F_n(r_i,s_j,t_k)}{\partial t} \tag{3.4}$$

STEP 2: JACOBIAN MATRIX. The [3 x 3] Jacobian matrix involves the partial derivatives of the element coordinates with respect to the intrinsic coordinates and the node point coordinates. Each matrix term requires an inner product of length N

$$J_{ij} = \sum_n \frac{\partial F_n}{\partial u_i} w_j \quad \text{where } u_i \leftarrow r,s,t \text{ and } w_j \leftarrow x,y,z \tag{3.5}$$

STEP 3: DETERMINANT OF JACOBIAN MATRIX. The determinant of the Jacobian matrix enters eq. (3.2) as a multiplicative factor.

STEP 4: INVERSE OF JACOBIAN MATRIX. The inverse Jacobian, denoted $[\bar{J}]$, is needed to compute the strain-displacement matrix.

STEP 5: STRAIN-DISPLACEMENT MATRIX For each Gauss point a [6 x 3N] strain-displacement matrix has to be computed, but only 9 x N terms are nonzero, and for standard isoparametric theory only 3 x N of these are distinct

$$\begin{aligned}
\frac{\partial F_n}{\partial x} &= Fx_n = Fr_n \bar{J}_{11} + Fs_n \bar{J}_{12} + Ft_n \bar{J}_{13} \\
\frac{\partial F_n}{\partial y} &= Fy_n = Fr_n \bar{J}_{21} + Fs_n \bar{J}_{22} + Ft_n \bar{J}_{23} \\
\frac{\partial F_n}{\partial z} &= Fz_n = Fr_n \bar{J}_{31} + Fs_n \bar{J}_{32} + Ft_n \bar{J}_{33}
\end{aligned} \tag{3.6}$$

$$[B] = \begin{bmatrix} Fx_1 & 0 & 0 & Fx_2 & 0 & 0 & \dots & Fx_n & 0 & 0 \\ 0 & Fy_1 & 0 & 0 & Fy_2 & 0 & \dots & 0 & Fy_n & 0 \\ 0 & 0 & Fz_1 & 0 & 0 & Fz_2 & \dots & 0 & 0 & Fz_n \\ Fy_1 & Fx_1 & 0 & Fy_2 & Fx_2 & 0 & \dots & Fy_n & Fx_n & 0 \\ 0 & Fz_1 & Fy_1 & 0 & Fz_2 & Fy_2 & \dots & 0 & Fz_n & Fy_n \\ Fz_1 & 0 & Fx_1 & Fz_2 & 0 & Fx_2 & \dots & Fz_n & 0 & Fx_n \end{bmatrix} \quad (3.7)$$

STEP 6: FIRST MATRIX PRODUCT. For orthotropic or isotropic material the stress strain matrix is of the form

$$[D] = \begin{bmatrix} d_{11} & d_{12} & d_{13} & 0 & 0 & 0 \\ & d_{22} & d_{23} & 0 & 0 & 0 \\ & & d_{33} & 0 & 0 & 0 \\ & & & d_{44} & 0 & 0 \\ & & & & d_{55} & 0 \\ \text{symmetric} & & & & & d_{66} \end{bmatrix} \quad (3.8)$$

15 x N multiplications are required to perform the first matrix product. The resulting matrix [A] is of order [6 x 3N] with 15 x N nonzero terms.

$$[A] = [D]\,[B] \quad (3.9)$$

STEP 7: SECOND MATRIX PRODUCT. For each Gauss point a stiffness matrix contribution is obtained from

$$[K]_{el} = [B]^T\,[A] \quad (3.10)$$

If symmetry is exploited only the upper (or lower) triangle of the stiffness matrix is constructed. Let L = 1/2((N x M)^2 + (N x M)) denote the number of terms in the stiffness matrix triangle, then 3L multiplications and 2L additions are performed for each $[K]_{el}$.

In the case of isotropic material the sparsity of [A] may be exploited and the number of operations reduces to $15N + 21/2\ ((N-1)^2 + (N - 1))$ multiplications and $10N + 14/2((N - 1)^2 + (N - 1))$ additions for each element stiffness matrix contribution at a particular Gauss point.

STEP 8: ASSEMBLY OF ELEMENT STIFFNESS MATRIX. The final element stiffness matrix is obtained by adding the Gauss point contributions. The presubscript p emphasizes that different matrices have to be computed at each Gauss point, i.e. STEP 1-7 are repeated P times.

$$[K]_{el} = \sum_{p=1}^{P} {}_p[K]_{el} \qquad (3.11)$$

Hence, another L x P additions are necessary.

An operation count for the element stiffness matrix generation is given in Table 1 for an 8-noded hexahedronal solid element with a 2 x 2 x 2 integration scheme. It can be seen that STEP 7 is by far most expensive.

Step #	Multiplications	Additions
1	960	384
2	576	504
3	96	24
4	216	72
5	576	384
6	960	0
7	5664	3776
8	0	2400
Sum	9048	7544

Table 3.1. Operation Count for Element Stiffness Matrix Generation: Isoparametric Hexahedron with N = 8, P = 8; isotropic material.

3.3 MSC/NASTRAN TETRAHEDRONAL ELEMENT

The element stiffness formulation of MSC/NASTRAN's tetrahedronal element follows the above outlined standard isoparametric procedure. The element consists of four corner nodes and optional midside nodes on each edge (N = 4... 10). A five point integration scheme is used as a default but the user may specify one point integration.

Initially the element routine (ETETD) performs geometry checks, open core memory allocations and computes the element-to-basic transformation matrix. The grid point coordinates are transformed to element coordinates in which the element stiffness matrix is generated. These preprocessing tasks are denoted as STEP 0.

STEPS 1-5 are carried out in the utility routine SHAPED (SHAPED is used for all MSC/NASTRAN solid elements). The strain displacement matrix, the determinant and inverse Jacobian are passed back to ETETD. All Gauss points are processed and the data are stored in open core memory before ETETD proceeds with STEP 6.

STEPS 6-8 are combined in a double loop structure, the outer loop is over all integration points, the inner loop is over all node points. The element stiffness matrix is computed for one Gauss point at a time. Provisions are made in the code to take advantage of the sparsity in the [A] matrix for the case of isotropic (or orthotropic) material. It should be noted that in case of temperature dependent material properties the stress strain matrix may be different for each integration point (call MAT inside the loop).

The element stiffness matrix is computed in terms of nodal partitions, the data structure, however, represents a columnwise storage scheme of the entire matrix in form of a one-dimensional field in open core memory. Rows and columns corresponding to deleted nodes are omitted. This data structure is a MSC/NASTRAN standard, all element routines have to generate their element matrices in this format. Before exit ETET makes a call to utility routine EMGPOM which writes the element stiffness matrix to the appropriate NASTRAN file.

Step #	Multiplications	Additions
0	ca. 200	ca. 200
1	435	260
2	450	450
3 & 4	155	45
5	405	300
6	750	0
7 & 8	5475	5475
Sum	8010	6780

Table 3.2. Operation Count for Element Stiffness Matrix Generation: Isoparametric Tetrahedron with N = 10, P = 5; isotropic material.

An operation count for the element stiffness matrix generation is given in Table 3.2 for a 10-noded tetrahedronal solid element with a five point integration scheme. The operations in the preprocessing phase are approximate values which do not account for initialization tasks. As seen previously STEP 7 is by far most expensive.

Vectorization of element computations requires special consideration in terms of code design. An inspection of the current TETRA code reveals that some effort has been taken to minimize computations and storage requirements but no provisions have been made to take advantage of vector processing. The length of computer vectors which are processed in vectorizable loops rarely exceeds 10 words and the code design inhibits vectorization in the most crucial steps.

3.4 VECTORIZED TETRA: SINGLE ELEMENT VECTORIZATION

In the following a highly vectorized procedure for the element stiffness matrix generation is outlined on the basis of a single element. The approach taken here achieves vectorization by processing Gauss points in parallel and by using vector replication schemes for the crucial matrix multiplications. The proposed code has been incorporated in an experimental version of the TETRA routine ETET. A system flag is passed into ETET to indicate vector processing.

Step 0 of the current ETET routine is maintained without changes for the vectorized code. A new shape function routine VSHAP has been developed which comprises STEPS 1-5 as described in the following. This approach is particularly attractive because a vectorized family of solid elements can be developed by using VSHAP for HEXA and PENTA elements as well. Also, on scalar machines, the current routine SHAPE could be maintained to avoid the replication overhead needed for vectorization.

STEP 1: SHAPE FUNCTION DERIVATIVES. The shape functions of tetrahedronal elements are not computationally equivalent at different node points, also their derivatives are not easily obtained with vector operations. However, the computational expense in STEP 1 can be reduced drastically by recognizing that the same derivative functions can be used for different elements if they have an equivalent node map. For most FEM models it can be expected that there is a large number of equivalent elements (e.g. ten-noded TETRAs) and only a small number of transitional elements with varying node numbers.

A simple modification to the code is proposed which does not affect the flexibility of the current NASTRAN elements. After generating the shape function derivatives for the first element a flag array is set which indicates the currently active nodes, the next element (which is of same element type!) checks this flag. If it has an equivalent node map it may use the shape function derivatives which are still stored in open core, hence STEP 1 is performed only once for a large group of elements and lack of vectorization does not affect the overall performance. For the following steps, however, it is crucial that the shape function derivatives are organized as three computer vectors of length N x P, (inactive node points are deleted!).

$$\vec{f}_r = [Fr_1 \ldots Fr_N|_1 \ldots |_{P-1} Fr_1 \ldots Fr_N|_P]$$

$$\vec{f}_s = [Fs_1 \ldots Fs_N|_1 \ldots |_{P-1} Fs_1 \ldots Fs_N|_P] \qquad (3.12)$$

$$\vec{f}_t = [Ft_1 \ldots Ft_N|_1 \ldots |_{P-1} Ft_1 \ldots Ft_N|_P]$$

STEP 2: JACOBIAN MATRIX. The inner product of eq. (3.5) can be stringed for all the Gauss points to obtain a vector length of N x P. The three vectors of nodal coordinates are replicated P times.

$$\begin{aligned}
\vec{x} &= [x_1 \ \dots \ x_N|_1 \ \dots \ |_{P-1} \ x_1 \ \dots \ x_N|_P \ ] \\
\vec{y} &= [y_1 \ \dots \ y_N|_1 \ \dots \ |_{P-1} \ y_1 \ \dots \ y_N|_P \ ] \\
\vec{z} &= [z_1 \ \dots \ z_N|_1 \ \dots \ |_{P-1} \ z_1 \ \dots \ z_N|_P \ ]
\end{aligned} \tag{3.13}$$

The terms of the Jacobian matrix are obtained in parallel for all Gauss points.

$$\begin{aligned}
\vec{j}_{11} &= \vec{x} \cdot \vec{f}_r \ , \quad \vec{j}_{12} = \vec{y} \cdot \vec{f}_r \ , \quad \vec{j}_{13} = \vec{z} \cdot \vec{f}_r \\
\vec{j}_{21} &= \vec{x} \cdot \vec{f}_s \ , \quad \vec{j}_{22} = \vec{y} \cdot \vec{f}_s \ , \quad \vec{j}_{23} = \vec{z} \cdot \vec{f}_s \\
\vec{j}_{31} &= \vec{x} \cdot \vec{f}_t \ , \quad \vec{j}_{32} = \vec{y} \cdot \vec{f}_t \ , \quad \vec{j}_{33} = \vec{z} \cdot \vec{f}_t
\end{aligned} \tag{3.14}$$

The dot in eq. (3.14) and in the following denotes the scalar product of two computer vectors (i.e. an inner product without summation of terms). Contracting the nine vectors yields the nine matrix terms at P integration points.

$$_pJ_{ij} = \sum \vec{j}_{ij} \ ; \quad p = 1,\dots,P \ ; \quad i,j = 1,2,3 \tag{3.15}$$

STEP 3: DETERMINANT OF JACOBIAN MATRIX. According to Tables 1 and 2 the computational expense for the determinant of a [3 x 3] matrix is small compared to the other steps and vectorization may be omitted.

STEP 4: INVERSE OF JACOBIAN MATRIX. Also the inverse of the Jacobian does not contribute much computational expense, for the following steps, however, it is important that the terms of the inverse Jacobian are organized as proper vectors. The multiplicative weight factors and the determinant of the Jacobian are conveniently incorporated here. Nine vectors $\vec{j}_{11}, \vec{j}_{12} \dots \vec{j}_{33}$ are set up. The terms of the weighted inverse Jacobian are replicated to obtain a vector length N x P.

$$\vec{j}_{ij} = [J_{ij} \ldots J_{ij}|_1 \ \ldots \ |_{P-1} J_{ij} \ldots J_{ij}|_P]; \ i,j = 1,2,3 \qquad (3.16)$$

STEP 5: STRAIN-DISPLACEMENT MATRIX. The nonzero terms of the strain-displacement matrix are computed concurrently at all Gauss points in the form of three computer vectors.

$$\begin{aligned}
\vec{b}_x &= [Fx_1 \ldots Fx_N|_1 \ldots |_{P-1} Fx_1 \ldots Fx_N|_P] \\
\vec{b}_y &= [Fy_1 \ldots Fy_N|_1 \ldots |_{P-1} Fy_1 \ldots Fy_N|_P] \qquad (3.17) \\
\vec{b}_z &= [Fz_1 \ldots Fz_N|_1 \ldots |_{P-1} Fz_1 \ldots Fz_N|_P]
\end{aligned}$$

The vector $\vec{b}_x$ for example is computed from

$$\vec{b}_x = \vec{j}_{11} \cdot \vec{f}_r + \vec{j}_{12} \cdot \vec{f}_s + \vec{j}_{13} \cdot \vec{f}_t \qquad (3.18)$$

Here a total of 9 vector multiplications and 6 vector additions each of length N x P are performed.

STEPS 6-8 are performed in ETET. If vector processing is requested the element stifness matrix is computed in VETET and the stiffness integration loops of ETET are skipped. VETET is a new routine which follows the design outlined below. Input is the strain-displacement matrix generated in VSHAP.

STEP 6: FIRST MATRIX PRODUCT. For the nonzero terms of the intermediate matrix [A] the following notation is used

$$[A] = \begin{bmatrix}
a01_1 & a02_1 & a03_1 & | \ \ldots \ | & a01_N & a02_N & a03_N \\
a04_1 & a05_1 & a06_1 & | \ \ldots \ | & a04_N & a05_N & a06_N \\
a07_1 & a08_1 & a09_1 & | \ \ldots \ | & a07_N & a08_N & a09_N \\
a10_1 & a11_1 & 0 & | \ \ldots \ | & a10_N & a11_N & 0 \\
0 & a12_1 & a13_1 & | \ \ldots \ | & 0 & a12_N & a13_N \\
a14_1 & 0 & a15_1 & | \ \ldots \ | & a14_N & 0 & a15_N
\end{bmatrix} \qquad (3.19)$$

The matrix terms are grouped in 15 vectors of length N x P, i.e. each vector contains N matrix terms repeated for the P Gauss points.

$$\vec{a}l = [al_1 \ldots al_N|_1 \ldots |_{P-1} al_1 \ldots al_N|_P] \ ; \ \ l = 1,\ldots,15 \qquad (3.20)$$

No vector replication is necessary, the $\vec{a}1$ are computed from a simple scalar-vector multiplication.

$$\vec{a}01 = d_{11}\ \vec{b}_x \ \ldots\ \vec{a}15 = d_{66}\ \vec{b}_x \tag{3.21}$$

In the base of anisotropic material the [A] matrix is full. Now 18 vectors of length N x P are computed.

$$\vec{a}01 = d_{11}\ \vec{b}_x + d_{14}\ \vec{b}_y + d_{16}\ \vec{b}_z \ \ldots\ \vec{a}18 = d_{63}\ \vec{b}_z + d_{65}\ \vec{b}_y + d_{66}\ \vec{b}_x \tag{3.22}$$

The computational expense for STEP 6 increases more than threefold, the payoff due to vector processing is higher than for isotropic material.

STEP 7: SECOND MATRIX PRODUCT This step is computationally most expensive and high payoffs are possible by proper vectorization. Special data structures are employed to achieve efficient vector processing. The stiffness matrix is generated in terms of nodal partitions and some redundant matrix terms are computed in the diagonal partitions. A final step has to be added to obtain the element stiffness matrix in the standard NASTRAN format (see above). A vector replication scheme is proposed to achieve a vector length of 1/2 (N^2 + N) x P which is considerably larger than the N x P vectors of the previous steps. It is believed that the introduced overhead is small compared to the gains in computational efficiency. The element stiffness matrix is formed in terms of R nodal partitions, see eq. (3.26)

$$R = 1/2(N^2 + N) \tag{3.23}$$

The stress-strain matrix was previously obtained in the form of three vectors $\vec{b}_x$, $\vec{b}_y$, $\vec{b}_z$. These vectors are partially replicated and concatenated for all Gauss points. Three vectors, denoted $\vec{\omega}x$, $\vec{\beta}$, $\vec{\beta}x$ each of length R x P are set up, e.g.

$$\vec{\beta}x = \left\{ \begin{array}{cccccccc} bx_1 & \ldots & bx_1 & bx_2 & \ldots & bx_2 & \ldots & bx_N|_1 \\ & & & & \ldots & \ldots & & \\ bx_1 & \ldots & bx_1 & bx_2 & \ldots & bx_1 & \ldots & bx_N|_P \end{array} \right\} \tag{3.24}$$

The $\vec{a}l$ vectors are replicated in a slightly different way, 15 vectors are constructed (for anisotropic material a total of 18 vectors is set up).

$$\vec{\alpha}l = \left\{ \begin{array}{c} a1_1 \ldots a1_N \mid a1_2 \ldots a1_N \mid \ldots \mid a1_N|_1 \\ \ldots \quad \ldots \\ a1_1 \ldots a1_N \mid a1_2 \ldots a1_N \mid \ldots \mid a1_N|_P \end{array} \right\} \; ; \; l = 1,\ldots 15 \qquad (3.25)$$

A total of 18R x P terms have to be replicated which may introduce a serious overhead. Efficient replications may be performed by using 18 P vectorized gather operations on R long vectors. Only two different index vectors are needed if they are used with appropriate offsets.

In the case of N = 8, 36 submatrices have to be computed and the element stiffness matrix is obtained in the following format

$$[K]_{el} = \begin{bmatrix} K_{01} & K_{02} & K_{03} & K_{04} & K_{05} & K_{06} & K_{07} & K_{08} \\ & K_{09} & K_{10} & K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ & & K_{16} & K_{17} & K_{18} & K_{19} & K_{20} & K_{21} \\ & & & K_{22} & K_{23} & K_{24} & K_{25} & K_{26} \\ & & & & K_{27} & K_{28} & K_{29} & K_{30} \\ & & & & & K_{31} & K_{32} & K_{33} \\ & & & & & & K_{34} & K_{35} \\ & \text{Symmetric} & & & & & & K_{36} \end{bmatrix} \qquad (3.26)$$

The submatrices in eq. (3.26) are of order M as shown in eq. (3.27). The subscript l denotes the nodal partition and the superscript refers to the 9 terms in each submatrix.

$$[K_l]_{el} = \begin{bmatrix} k_l^1 & k_l^2 & k_l^3 \\ k_l^4 & k_l^5 & k_l^3 \\ k_l^7 & k_l^8 & k_l^9 \end{bmatrix} \quad ; \; l = 1,\ldots,36 \qquad (3.27)$$

The stiffness matrix contributions are organized in the form of nine vectors of length R x P. Each vector represents one term of all R submatrices over all Gauss points.

$$ {}_p\vec{k}^m = [k_1^m \ldots k_R^m \,|_1 \ \ldots \ |_{P-1} k_1^m \ldots k_R^m \,|_P]; \; m = 1,\ldots,9 \qquad (3.28) $$

These computer vectors are generated individually and again the sparsity of the matrices [A], [B] can be exploited for the nondiagonal terms in each submatrix.

$$ \begin{aligned} {}_p\vec{k}^1 &= \vec{\beta}x \cdot \vec{\alpha}01 + \vec{\beta}y \cdot \vec{\alpha}10 + \vec{\beta}z \cdot \vec{\alpha}12 \\ {}_p\vec{k}^2 &= \vec{\beta}x \cdot \vec{\alpha}02 + \vec{\beta}y \cdot \vec{\alpha}11 \\ &\ldots \quad \ldots. \\ {}_p\vec{k}^9 &= \vec{\beta}z \cdot \vec{\alpha}09 + \vec{\beta}y \cdot \vec{\alpha}13 + \vec{\beta}x \cdot \vec{\alpha}15 \end{aligned} \qquad (3.29) $$

A certain overhead, however, is inherent to this procedure. The N submatrices on the diagonal are treated the same way as the nondiagonal submatrices, i.e. nine matrix terms are computed but three of these are redundant (symmetry!). Compared to the scalar code N x 3 x P additional terms are introduced resulting in an overhead of N x 6 x P multiplications and N x 3 x P additions.

This overhead could be avoided by separating the diagonal and nondiagonal submatrices in two groups, but the vector lengths would be reduced to N x P and 1/2 $((N - 1)^2 + N - 1)$ x P respectively.

STEP 8: ASSEMBLY OF ELEMENT STIFFNESS MATRIX. Contracting the nine ${}_p\vec{k}^m$ for the Gauss points with P vector additions of length R renders nine final vectors.

$$ \vec{k}^m = \sum_{p=1}^{P} {}_p\vec{k}^m \qquad (3.30) $$

The first submatrix corresponds to the first entry in $\vec{k}^m$, the second submatrix to the second entry and so on.

At this stage the element stiffness matrix is fully generated. The following STEP 9 is necessary to reorganize the data into the NASTRAN format. It should be noted, however, that submatrix format could be used effectively to transform the element stiffness matrix from element coordinates to basic coordinates. All terms in the $\vec{k}^m$ vectors are to be multiplied with one corresponding term in the element-to-basic transformation matrix. 27 scalar times vector operations and 18 vector additions are performed on vectors of length R to accomplish the transformation.

STEP 9: DATA SHUFFLE. The NASTRAN element stiffness matrices are stored as a pseudo vector Z which represents the active rows of the upper triangle (or the active columns of the lower triangle). Consecutive rows are concatenated and the field length of Z is $L = 1/2((N \times M)^2 + (N \times M))$. This data structure can be obtained from the $\vec{k}^m$ vectors by using vector scatter operations. Nine index vectors have to be set up. The first index vector $\vec{ix}^1$ maps the stiffness matrix terms $\vec{k}^1$ to the Z array and so on, nine vector scatters of length R are required. Care has to be taken that the redundant matrix terms (see SETP 7) are mapped to a dummy address and not into the Z array.

A summary of the vectorized procedure is given in Table 3.3 for a 10-noded element using a five point integration scheme. Isotropic material and no temperature variations throughout the element are assumed.

No. of Operations/Vector Length			
Step #	Vector Addition	Vector Multiplication	Gather/Scatter
1-5	9/50	12/50	12/50
6	----	15/50	----
7	12/275	21/275	18/275
8	45/55	---	----
9	----	---	9/55

Table 3.3 Vectorized Element Stiffness Matrix Generation: Isoparametric Tetrahedron with N = 10, P. = 5; isotropic material.

3.5 VECTORIZED TETRA: MULTIPLE ELEMENT VECTORIZATION

Vectorizing element computations on the basis of multiple elements is conceptually simple. Instead of generating the stiffness matrix for each element individually, the information is obtained for a group of NEL elements in parallel (the same concept may be applied to element stresses and force vectors). For example, an individual datum (e.g. an element stiffness matrix term at a particular Gauss point) is computed for all NEL elements; the natural data structure is an array of length NEL.

NOTE: In this context processing elements in "parallel" does not refer to multi-processing. The data are generated conceptually in parallel but not concurrently. However, it is quite obvious that processing elements in parallel can be used for vector and multi-processing.

The process of vectorization is best demonstrated with a simple example. Consider STEP 6 of the element stiffness generation. Evaluating equation (3.9) at a particular integration point of an individual element yields 15 N expressions (compare with equations (3.7,3.8,3.19) for notation):

$$\begin{aligned} a01_1 &= d_{11}\, Fx_1 \\ &\cdots \quad \cdots \\ a15_N &= d_{66}\, Fz_N \end{aligned} \tag{3.31}$$

Equation (31) can be evaluated for NEL elements in parallel with a simple loop structure which will be recognized as vectorizable by any vectorizer compiler.

```
do for i = 1, NEL
   a01(i)_1 = d_11 Fx(i)_1
    ...          ...
   a15(i)_N = d_66 Fz(i)_N
end do
```

(3.32)

This example shows the ease with which multiple element vectorization can be incorporated in conventional code. The vector length is constant throughout the whole procedure and simply governed by NEL, the loop stride is equal to one.

The above procedure can be applied to STEPS 2-8 of the element stiffness generation. STEP 1, the computation of shape function derivatives, is performed only once for the entire element group and is therefore computationally insignificant. Analogons to the single element vectorization a final step has to be added to recast the matrices into the standard NASTRAN format (compare with STEP 9 above). After STEP 8 the matrix terms of an individual element do not occupy a continuous field in memory but they are separated by NEL words belonging to the other element matrices. All matrix terms are stored in a single array of length NEL x L, where L is the size of one element stiffness matrix, and the reformatting requires one scatter (or gather) operation on this array.

For optimal computational efficiency it is desirable to select NEL such that the vector registers are filled completely but the method does work (with varying efficiency) for any value of NEL. Limits may be imposed on NEL by

- the number of available computationally equivalent elements and
- the amount of available high speed memory.

Only computationally equivalent elements may be processed in parallel. For example, the vectorized element group may comprise only elements of the same type (e.g. TETRA), same node map (e.g. 10-noded), same material type (e.g. isotropic) and same number of integration points. The vector register length for most computers falls in the range of 32-128 (double precision) words. Complex structural problems, however, are modeled with thousands of finite elements most of which will be computationally equivalent in above sense. Hence, the availability of sufficient elements does not restrict multiple element vectorization in most cases.

The amount of required memory is equal to 2 x NEL x L; the factor 2 is due to the reformatting of the element stiffness matrices described above because both the original and reformatted arrays have to be kept in memory. All other arrays which are needed during STEPS 2-8, e.g. a01(i) of equation (3.32), can be equivalenced into the additional stiffness matrix field and no further memory is needed.

A vectorized software testbed for multiple TETRA elements has been developed on the basis of current NASTRAN code by using the procedure outlined in equation (3.32). Only STEPS 6-8 have been vectorized in the version presented here. The basic double loop structure with the outer loop over all node points and the inner loop over all integration points remained unchanged because this approach is most efficient in terms of memory requirements. To save memory in a final version the computation of shape functions and their derivatives should also be incorporated in this structure such that data are generated separately for all Gauss points.

Reformatting the element stiffness matrices into the standard NASTRAN pseudo column format is done in STEP 9 by using vector scattering (see above).

3.6 VECTORIZED TETRA: PERFORMANCE DATA AND NASTRAN IMPLEMENTATION

The element stiffness matrix generation in NASTRAN is performed in the EMG module. The module is currently designed such that elements are processed individually with an outer loop over all elements types and an inner loop over all elements of one particular type. This structure favors single element vectorization which was therefore tested in detail. A vectorized TETRA element has been developed at Georgia Tech and was subsequently incorporated in MSC/NASTRAN. The experimental code is currently restricted to the linear elastic case. Initially the code was implemented on the MSC VAX cluster to validate the code, performance tests were run on a CONVEX C-1 computer.

To establish some baseline data the current TETRA code was recompiled with and without compiler vectorization. A simple test problem composed of 1000 10-noded TETRA elements was used and detailed timing results were obtained. As expected no speedup could be achieved by simply recompiling the current routine with the vectorizer compiler. The execution times of the current and the vectorized element are combined in Table 3.4. Three different phases of the procedure were measured: preface (STEP 0), stiffness matrix generation (STEPS 1-9) and writing the matrices to the disk.

Program Phase	Current TETRA (CPU sec.)	Vectorized TETRA (CPU sec.)	Speedup (Scalar/vector)
Preface	1.5	1.5	1.0
El. Stiff. Matrix	11.5	8.9	1.2
Write	11.5	11.5	1.0
Total	24.8	22.6	1.1

Table 3.4 Comparison of Current TETRA and Vectorized TETRA on CONVEX: Single Element Vectorization, 1000 Elements.

An unexpected result was the high amount of CPU time required to perform the I/O operations. Although not the subject of this study, this result is important because it severely limits the possible gains in execution time by vectorizing NASTRAN's EMG module. At this point it is not clear whether this is a CONVEX related phenomenon or whether these results are typical for NASTRAN also on other machines. In the following only in-core operations are discussed.

Comparing single element vectorization with scalar element processing it was found that only a low speedup of 1.2 could be achieved. A more detailed performance assessment was obtained to explain the relatively poor performance. In Table 3.5 the CPU time for the individual steps of the matrix generation are evaluated.

Step	Current TETRA (CPU sec.)	Vectorized TETRA (CPU sec.)	Speedup (scalar/vector)
STEPS 1-5	2.8	2.0	1.4
STEPS 6-9	8.7	6.9	1.2
Total	13.0	10.4	1.2
Gather for 7	---	2.3	---
STEP 9	---	.42	---

Table 3.5 Comparison of Current TETRA and Vectorized TETRA on CONVEX.: Single Element Vectorization, 1000 Elements.

As expected from the operation count STEPS 6-9 are the computationally most expensive task for the current TETRA code. The speedup for the vectorized code is only 1.2 which is partially due to the gather/scatter overhead (see the last two rows of Table 3.5). However, if we subtract this overhead, the STEPS 6-8 still take more than half of the scalar code (a speedup of 2). This result is not yet well understood, since from the machine characteristics (vector/scalar speed ratio) a factor of at least 4 was expected.

Since the performance characteristic of any vetorized code is to a certain extent machine dependent, STEPS 6-9 of both the current TETRA element and the vectorized version were implemented on a CRAY X-MP Computer at the Pittsburgh Supercomputer Center in a standalone environment. Table 3.6 shows the measured CPU times.

Step	Current TETRA (CPU sec.)	Vectorized TETRA (CPU sec.)	Speedup (scalar/vector)
STEPS 6-8	0.79	0.19	4.2
Gather for 7	---	0.21	---
STEP 9	---	0.05	---
Total	0.79	0.45	1.8

Table 3.6 Comparison of Current TETRA and Vectorized TETRA on CRAY X-MP: Single Element Vectorization, 1000 Elements, STEPS 6-9 Only.

Two observations can be made. The overall performance of the vectorized code is significantly better on the CRAY than on the CONVEX, and if the overhead is subtracted the actual matrix computation time is reduced from 0.792 CPU sec. to 0.191 CPU sec. Inspite of the better performance it can be seen that on the CRAY the overhead to set up the computer vectors takes actually longer than the "useful" computations. This is due to the fact that vectorized gather/scatter operations are almost as time consuming as vector additions or multiplications. Hence, the replication overhead of the single element vectorization is significant compared to the increased speed of vectorized computations.

The multiple element procedure described above has been implemented also on the CRAY X-MP. The same TETRA element was chosen but only STEPS 6-9 were available for this study. The maximum number of parallel elements was chosen as 64 and 1000 elements were processed. The overall execution time is given in Table 3.7 together with speedup and memory requirements. For comparison the corresponding data of the scalar and the vectorized single element procedures are compiled in the same table.

Code	Current TETRA	Vectorized Single TETRA	Vectorized Multiple TETRA
CPU Time (sec.)	0.79	0.45	0.15
Speedup	1.0	1.8	5.2
Memory (Words)	540	7,425	59,520

Table 3.7 Comparison of Current TETRA and Vectorized TETRA on CRAY X-MP: 1000 Elements, STEPS 6-9 Only.

The multiple element method shows superior performance with a speedup of 5.2. The penalty is the drastically increased memory requirement to store 2 x 64 element matrices simultaneously. Going from scalar code to single element vectorization also increases the memory requirements but 7000 Words are not significant compared to the typically available memory.

Implementing vectorized element procedures in MSC/NASTRAN effects different levels of recoding and system design. The implementation of a single element vectorization TETRA element has been completed during the course of this study. The element routines have to be recoded and due to the overhead vectorized and nonvectorized versions of the elements are needed for vector and scalar machines. No changes are necessary in terms of system design, the implementation can be performed without any changes in the current EMG module.

The multiple element vectorization method also requires major modifications on the element level. The changes, however, appear to be relatively straight forward since the basic routine design remains unchanged. But major modifications have to be made on the module level. Since elements are processed in groups input and output of element data has to be performed in groups as well. Currently the input data (e.g. the EST table) and the output data (the element matrix) and dictionary are processed element by element. Furthermore provisions have to be made to identify groups of computationally equivalent elements. No changes would occur on higher than the module level.

An important aspect for the feasibility of multiple element vectorization are the increased memory requirements. Currently the 20-noded HEXA is the most complex element implemented in NASTRAN in terms of memory requirements. The stiffness matrix of such an element requires 1860 words of memory, hence, a field of 234,240 words is needed for a multiple element procedure. This can be taken as an upper bound for the required open core memory because a recoded EMG module would not require any additional fields. The strict separation of element matrix generation (EMG) and element matrix assembly (EMA) in NASTRAN is a valuable asset because all the memory is available for element stiffness matrices.

3.7 CONCLUSIONS AND RECOMMENDED FUTURE WORK

It has been shown that both single element vectorization and multiple element vectorization can be used to reduce the computational expense of element related computations in MSC/NASTRAN. Performance data has been presented comparing the current TETRA element with vectorized procedures. The multiple element method yields better computational efficiency but significant changes in the EMG module are necessary to implement such a procedure in NASTRAN. It has been discovered that the I/O operations during the EMG phase are computationally very expensive (CPU time spent for I/O!).

In view of upcoming highly vectorized FEM systems [3.6-3.10] further performance improvements have to be incorporated in MSC/NASTRAN. These performance improvements will have to come from element dependent procedures because NASTRAN's matrix modules are already well vectorized.

It is recommended that a multiple element vectorization method be considered for implementation in NASTRAN. An experimental EMG module could be designed without changes to the data base or the executive system. The double loop structure of the EMG module could be used effectively. Only one element (e.g. TETRA) would have to be recoded for preliminary performance tests. A design sketch is given below.

Leave the (outer) element type loop of EMG unchanged but redesign the (inner) element loop using the following approach:

- Read and store the EST table of the first element of current type.
- Read and store the EST table of the next element of same type.
- Check if elements are computationally equivalent, if yes proceed.
- If no or if enough elements gathered, generate element matrices.
- Pack out the element matrices of the group and proceed.

In the course of the proposed redesign the current I/O operations should be screened, since savings of I/O expenses must be a major concern for future NASTRAN releases. Performing I/O operations on blocks of elements instead of individual elements as proposed above could prove to be an asset for reducing I/O expenses. The code design should have enough flexibility to also take advantage of concurrent vector computer architectures (see Section IV).

In this study only standard isoparametric elements like the TETRA have been considered. Many NASTRAN elements, however, do not follow the standard theory (e.g. HEXA, QUAD4) but use reduced shear techniques and internal strain functions. A feasibility study should be conducted to investigate multiple element methods for these elements.

Ultimately other matrices than linear elastic stiffness (differential stiffness, mass matrix, etc.) have to be included in the vectorized procedure. Nonlinear element force vector generation is another area which could benefit tremendously from vectorization (DYNA and NIKE codes already have such capabilities)..

3.8 REFERENCES

3.1 A. K. Noor, S. J. Hartley, "Evaluation of Element Stiffness Matrices on CDC STAR-100 Computer", Computers and Structures, Vol. 9, No. 2, 1978, pp. 151-161.

3.2 A. K. Noor, J. J. Lambiotte, "Finite Element Dynamic Analysis on CDC STAR-100 Computer", Computers and Structures, Vol. 10, 1979, pp. 7-19.

3.3 A. K. Noor, J. M. Peters, "Element Stiffness Computations on CDC 205 Computer", Communications in Applied Numerical Methods, Vol. 2, 1986, pp. 317-328.

3.4 M. Kratz, "Some Aspects of Using Vector Computers for Finite Element Analysis", Proceedings of the International Conference on Parallel Computing held at the Freie Universitaet Berlin, September 1983, North Holland, Amsterdam, 1984, pp. 349-354.

3.5 L. P. Nolte, B. Schiek, "Vectorized High Precision Finite Elements with Applications to Nonlinear Shell Problems", Proceedings of the Second International Conference on Parallel Computing held at the Freie Universitaet Berlin, September 1985, North Holland, Amsterdam, 1986, pp. 305-310.

3.6 J. O. Hallquist, "NIKE2D - A Vectorized Implicit, Finite Deformation, Finite Element Code for Analyzing the Static and Dynamic Response of 2-D Solids, UCID-19677", University of California, 1982.

3.7 J. O. Hallquist, "NIKE3D - An Implicit, Finite Deformation, Finite Element Code for Analyzing the Static and Dynamic Response of 2-D Solids, UCID-18822", University of California, 1984.

3.8 J. O. Hallquist, D. J. Benson, "DYNA3D User's Manual", UCID-19592, Rev. 2, University of California, 1986.

3.9 T. J. R. Hughes, R. M. Ferencz, J. O. Hallquist, "Large-Scale Vectorized Implicit Calculations in Solid Mechaics on CRAY X-MP/48 Utilizing EBE Preconditioned Conjugate Gradients", Computer Methods in Applied Mechanics and Engineering, Vol. 61, 1987, pp. 215-248.

3.10 D. P. Flanagan, "Concurrent Vectorized Processing Techniques in Transient Dynamics", Abstracts of the 7th ASCE Engineering Mechanics Speciality Conference, Blacksburg Va., 1988.

SECTION IV

PROGRAM DESIGNS AND DATA MANAGEMENT REQUIREMENTS FOR PARALLEL PROCESSING

In this section program designs for a largely parallel MSC/NASTRAN static solution are outlined. The file handling in a parallel environment is discussed since parallel processing of element data has to be seen in the context of the employed data base. Design strategies for parallel FEM systems are discussed and the relevant literature is reviewed.

4.1. FUNDAMENTALS OF PARALLEL PROCESSING

Parallelism has been used to improve the effectiveness of computers since the earliest designs and can be applied to different levels of a particular job:

1. Job level parallelism.
2. Program level parallelism.
3. Instruction level parallelism.

For large scale analyses job level parallelism can be effectively employed by overlapping buffered input/output operations of one job with the execution of another job. Program level parallelism means that within one job sections of the code are independent of each other and could be executed in parallel on multiple processing units. Instruction level parallelism may be exploited by overlapped execution of scalar instructions and is used in many computers [4.1].

Job level parallelism as described above is already incorporated in MSC/NASTRAN and is therefore not investigated any further in this study. Instruction level parallelism is an interesting approach but with the advent of vector processors it seemed to be obsolete. Recently, however, the "very long instruction word" approach has been introduced [4.2] which may prove to be a powerful tool to speed-up the existing finite element codes. From the viewpoint of a finite element code designer, however, this approach does not require substantial consideration in terms of specific software development because parallelism is extracted on a very low level by special compilers.

Program level parallelism in contrast to the other approaches requires considerable efforts in terms of "designing parallel code". Some automated capabilities (i.e. compilers) exist to extract program level parallelism from standard sequential code, but "rethinking the solution method" and "recoding existing algorithms" are generally necessary to exploit the full potential of parallel processing.

The design of FEM systems taking advantage of multiple processing units (in the following denoted in short as "parallel computing" or "parallel processing") is the objective of the research effort described in this section. In the following a brief overview of parallel computers is given as far as the machine architecture has important implications for the design of parallel finite element codes. Parallel computers have independent processing units, hence a multiple instruction stream is operating on a multiple data stream (MIMD) [4.3]. The theoretical performance gain S_{th} can be obtained from Amdahl's law

$$S_{th} = \frac{1}{(1-m) + \frac{m}{p}} \qquad (4.1)$$

where m denotes the amount of originally sequential CPU time which can be executed on p parallel processors. Table 4.1 evaluates Equation (4.1) for selected values of p and m.

m	p = 4	p = 16	p = 64	p = 256
0.5	1.60	1.88	1.97	1.99
0.9	3.07	6.40	8.77	9.66
0.99	3.88	13.91	39.26	72.11

Table 4.1 Theoretical Performance Gain (Amdahl's Law).

It can be seen that for moderately parallel code (e.g. m = 0.5) using more than four processors does not result in any appreciable performance gains. Using 16 or more processors is only meaningful if more than 90% of the code runs in parallel. For massively parallel computing environments (e.g. 256 processors) more than 99% (!) of the job should be executed in parallel. The number of processors should be chosen according to the level of parallelism as shown in Table 4.2.

m	SUGGESTED NUMBER OF PROCESSORS
0.500	(MODERATELY PARALLEL) 2 - 4 CPU
0.900	(LARGELY PARALLEL) 4 - 16 CPU
0.990	(HIGHLY PARALLEL) 16 - 64 CPU
0.999	(MASSIVELY PARALLEL) 64 - 256 CPU

Table 4.2 Suggested Number of Parallel Processes for Different Levels of Parallelism.

Different computer architectures have been devised based on how memory is associated with the processors and how the processors communicate with each other. For the software designer it is useful to distinguish between local memory and shared memory architectures.

Local memory multiprocessors consist of a network of interconnected processors each having their own memory. Messages passing through the network is the only means of inter-processor communication. Typically 2^n processors are arranged in a hypercube configuration where each processor is connected to n other processors. The Intel iPSC and the N-Cube are important commerical machines in this calss [4.4].

Shared memory multi-processors consist of a number of processors and a large memory which can be directly accessed by any processor. Each processor may have seperate memory ports (e.g. the CRAY X-MP [4.5]), processors, memory may be connected with a bus system (e.g. ELXSI 6400 [4.6]) or a special switching network (e.g. Butterfly [4.7]). Some bus based systems provide local memory in addition to the shared memory thus eliminating the need to use the common bus for every memory reference (e.g. FLEX/32 [4.8]).

Some new supercomputers but also some remarkable "near supercomputers" have been designed as multi-level parallel machines, i.e. multiple processors are equiped with vector hardware. CRAY X-MP, IBM 3090/600 and the Alliant FX-8 are some important examples which belong to the class of shared memory machines, but also some local memory machines have been enhanced with vector hardware on each processor. The programming methodology is vectorization in the inner loops and multi-processing in the outer loops of a specific task, in many applications, however, a tradeoff between maximum vector length and multi-processing may arise.

Multipurpose FEM software systems commonly use a double level storage technique. Disk resident scratch files are used when the problem size exceeds the available central memory and intermediate results are stored for restart capabilities.

Migrating such software systems from uni-processors to a parallel computing environment may have important implications for data base design according to the chosen file handling strategy.

Disk resident files may be handled in three difference ways depending on the data base and the I/O capabilities of the particular multiprocessor:

1. The file access may be restricted to the main processor. Parallel processes operate only on memory resident data. By the nature of this approach no changes to the data base or the I/O utility routines are necessary (parallel code of this type has been successfully implemented in MSC/NASTRAN with the PDCOMP module). Since I/O is to be performed sequentially the level of parallelism may be limited if I/O requires a significant share of the total execution time.
2. Common files may be accessed from parallel processes as shown in Figure 4.3. The I/O utility routines have to be adopted for this purpose[1] (files are to be open for several processing units) but the data base may remain unaltered. Since the processes "talk" to the same files, physical I/O must be protected from concurrent execution, otherwise the data base may be corrupted. If the I/O time is substantially less

---

1. Modified GINO routines have been developed in Phase I of this work but due to hardware limitations they are not yet operational.

than the "truely parallel" code, each processor may use the I/O channel while the other processes carry out in-core computations. Processes may efficiently run in a cascaded time schedule. However, as more processes are added, an I/O bottle-neck will arise as illustrated in Figure 4.3

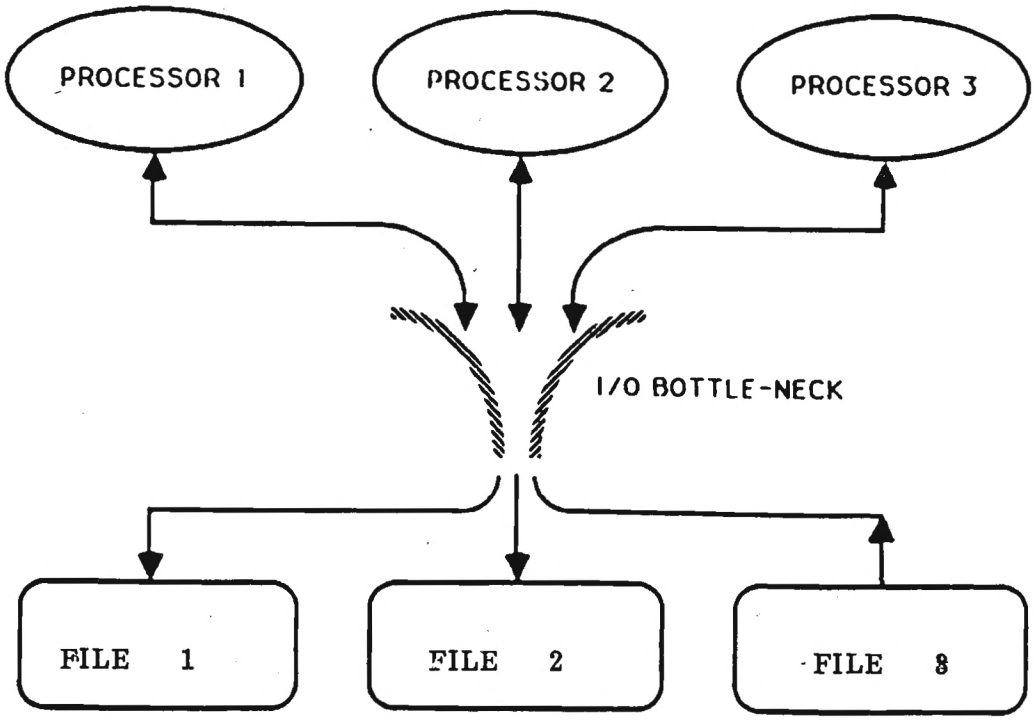


Figure 4.3. Parallel Processes Accessing Common Files.

3. Processes may work on seperate file systems using independent I/O devices as shown in Figure 4.4. This approach offers the highest potential performance but a distributed data base design is required. For example, the NASTRAN data base would have to be at least partially redesigned to accomodate independent file systems.

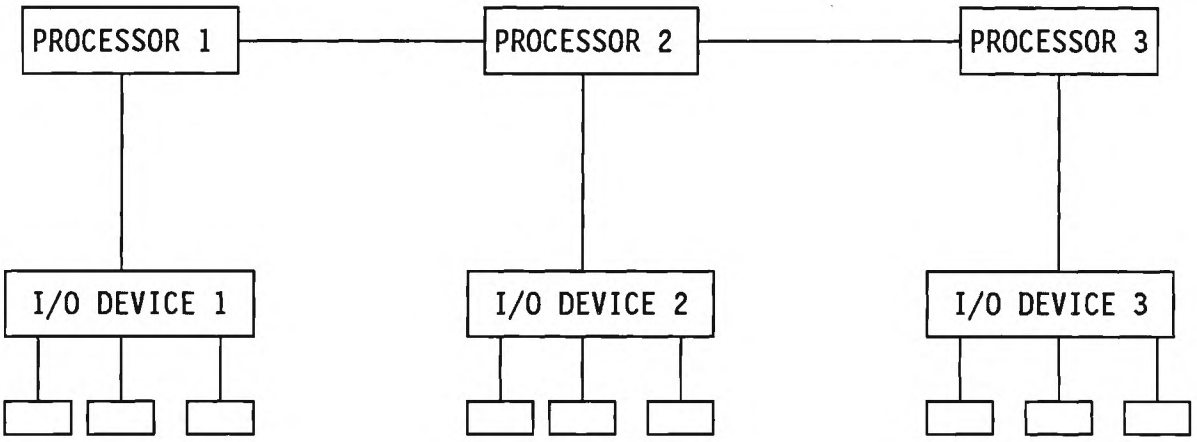


Figure 4.4. Parallel Processes and a Distributed Data Base.

4.2. LITERATURE REVIEW ON PARALLEL FEM PROCEDURES AND PROGRAM DESIGNS

Recently a respectable body of literature has emerged on parallel processing methods for finite element procedures. In the early 1980's first studies proposed the use of multi-processors for FEM analysis [4.41, 4.42], the need for a new finite element technology has been identified [4.43] and an ongoing multi-processing project has been established at NASA [4.44, 4.54]. An overview of parallelism in finite element modelling is given in [4.45] and parallel computer architectures for FEM analysis are discussed in [4.46].

Solving large sets of simultaneous equations is the most time consuming task in many finite element applications. Parallel Cholesky factorization algorithms for both message passing and shared memory multi-processors have been developed at the Oak Ridge National Laboratory [4.27, 4.28] and by others [4.29, 4.30]. Many studies in this field are of theoretical nature or merely present simulation results, several algorithmic variants have been tested on a hypercube machine [4.21] and noteworthy results have been achieved on a Butterfly computer with a Gaussian elimination procedure using up to 60 processors [4.32].

For large three-dimensional problems iterative solution techniques, such as Gauss-Seidel, SOR and conjugate gradient, can be more efficient than direct methods. Iterative methods to solve linear equations on multi-processors have been investigated even before such machines became available [4.17, 4.18]. Multi-processor implementations have been studied initially on the Finite Element Machine [4.19] and the Denelcor HEP system [4.20] and since then the literature on parallel iterative equation solvers has been growing rapidly [4.21-4.26].

A substantial amount of work has been undertaken on finite element methods built around the conjugate gradient algorithm [4.47-4.49]. This is because conjugate gradient methods can be organized in such a way that all computations are performed on the element level, no global matrices have to be assembled. The method is therefore well suited for local memory machines as well as shared memory computers.

Eigenvalue methods based on matrix decomposition procedures are well suited for parallel processing. Lanczos eigenvalue extractions have been implemented on vector processors [4.37] and on a FLEX/32 multi-processor [4.38], the efficiency of multi-processing mainly depends on the performance of the parallel decomposition procedure [4.39].

Transient dynamics problems may be solved with explicit integration schemes which have a high level of inherent parallelism. A study on the Finite Element Machine indicated the potential for significant speedup in solution times for FEM transient response problems [4.40].

Element evaluations, such as element stiffness matrix generation, are quite independent and may easily be distributed across multiple processors, but if a global stiffness matrix is to be formed, interprocess synchronization is necessary and memory contention has to be avoided. An implementation of the finite element method is under way on a Butterfly machine. Initial results indicate that the special switch architecture may avoid memory contention for the assembly of global matrices even for a large number of processors [4.49, 4.50].

Substructuring methods can be used for designing FEM methods suited for parallel processing by assigning parallel processors to seperate substructures. A first implementation on a local memory machine has been reported for linear analysis [4.51, 4.52] and a similar approach has been proposed for nonlinear problems [4.53].

4.3. LARGELY PARALLEL LINEAR STATIC SOLUTION IN MSC/NASTRAN

Linear static analysis of complex problems may require very large FEM models and hence substantial expenses of CPU time. The major computational work in MSC/NASTRAN is typically the triangular decomposition of the global stiffness matrix but other modules are also significant.

Table 4.5 lists the relative execution time for the major modules of a large statics problem. A standard MSC/NASTRAN test problem (BST 120) was used which has a total of 53,570 degrees of freedom. The problem was solved on a Convex C-1 machine in 3840 CPU sec.

MODULE	%CPU TIME
DCMP	70
EMA	8
EMG	4
FBS	4
SDR	4
Preprocessing	4
Constraints	3
Other	3

Table 4.5. Relative CPU Time for MSC/NASTRAN Modules.

In Version 66 of MSC/NASTRAN a parallel decomposition module (PDCOMP, developed jointly by L. Komzsik and D. Goehlich) and a parallel FBS (for multiple right hand sides) will be available. Hence the previous job could run 74% in parallel as shown in Table 4.6.

MODULE	% PARALLEL
Parallel DECOMP	70
Parallel FBS	4
MODERATELY PARALLEL	74

Table 4.6. Parallelism in MSC/NASTRAN Version 66 Static Solutions.

Hence, following the classification of Table 4.2, MSC/NASTRAN is moderatly parallel and the use of up to 4 CPU would result in appreciable wall clock time savings for the NASTRAN user.

In the near future, however, MSC/NASTRAN will become increasingly available on machines with up to 8 processors (ALIANT FX/8, CRAY Y-MP) and it is expected that future supercomputers will take advantage of even more processors. In the following we suggest some new code designs and data structures which could upgrade MSC/NASTRAN to a largely parallel system with about 90% parallism for large linear statics applications.

Using the previous test problem it can be deduced that EMG, EMA and SDR modules have to be executed in parallel to reach above goal (See Table 4.7)

MODULE	% PARALLEL
Parallel DECOMP Parallel FBS	70 4
Parallel EMG Parallel EMA Parallel SDR	4 8 4
LARGELY PARALLEL	90

Table 4.7. Proposed Largely Parallel MSC/NASTRAN.

4.3.1 Parallel EMG in MSC/NASTRAN

Two features of the current EMG module are important for a new parallel EMG module. Firstly the order of element processing with an outer loop over all element types and an inner loop over all elements. Secondly the element by element access (Read and Write) of the data base. EMGPRO is the actual *"workhorse"* of EMG which calls the appropriate element routines as shown in Figure 4.8. The element summary table (EST) is read in EMGPRO while the output data blocks (KELM, EDICT) are processed in the element routines (element routine calls EMGPOM).

Parallel processing in the EMG phase is conceptually simple. Different elements can be processed independently on multiple CPU if all the data remains memory resident. Figure 4.9 shows such an approach which could be implemented in MSC/NASTRAN in the framework of the current data base. EMGPRO would be modified in such a way that it reads the EST of N elements (Phase I, one processor only). The maximum number of memory resident elements N will be determined by

$$N = \frac{\text{size of open core}}{\text{words per element}} \qquad (4.2)$$

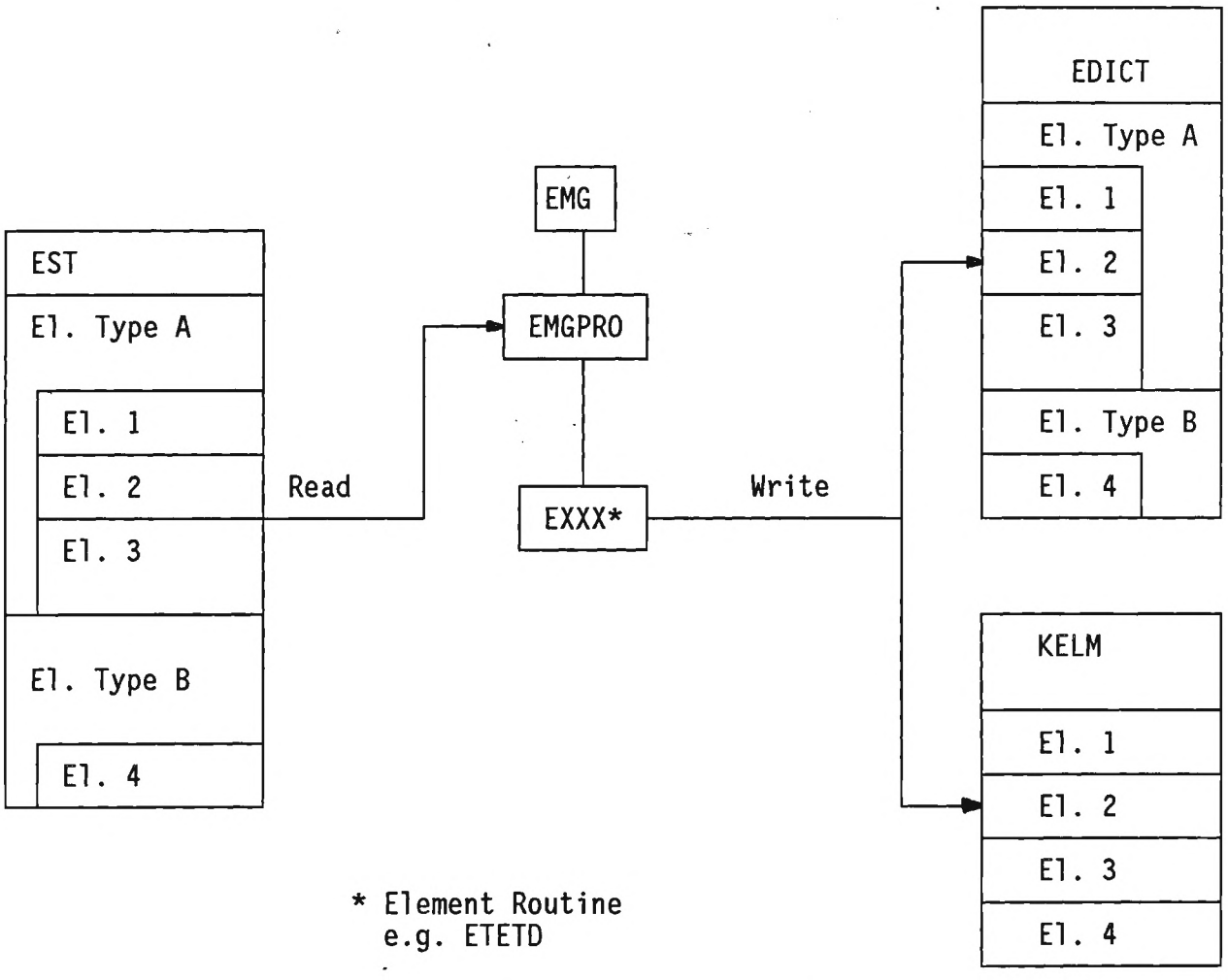


Figure 4.8. EMG Module Flowchart.

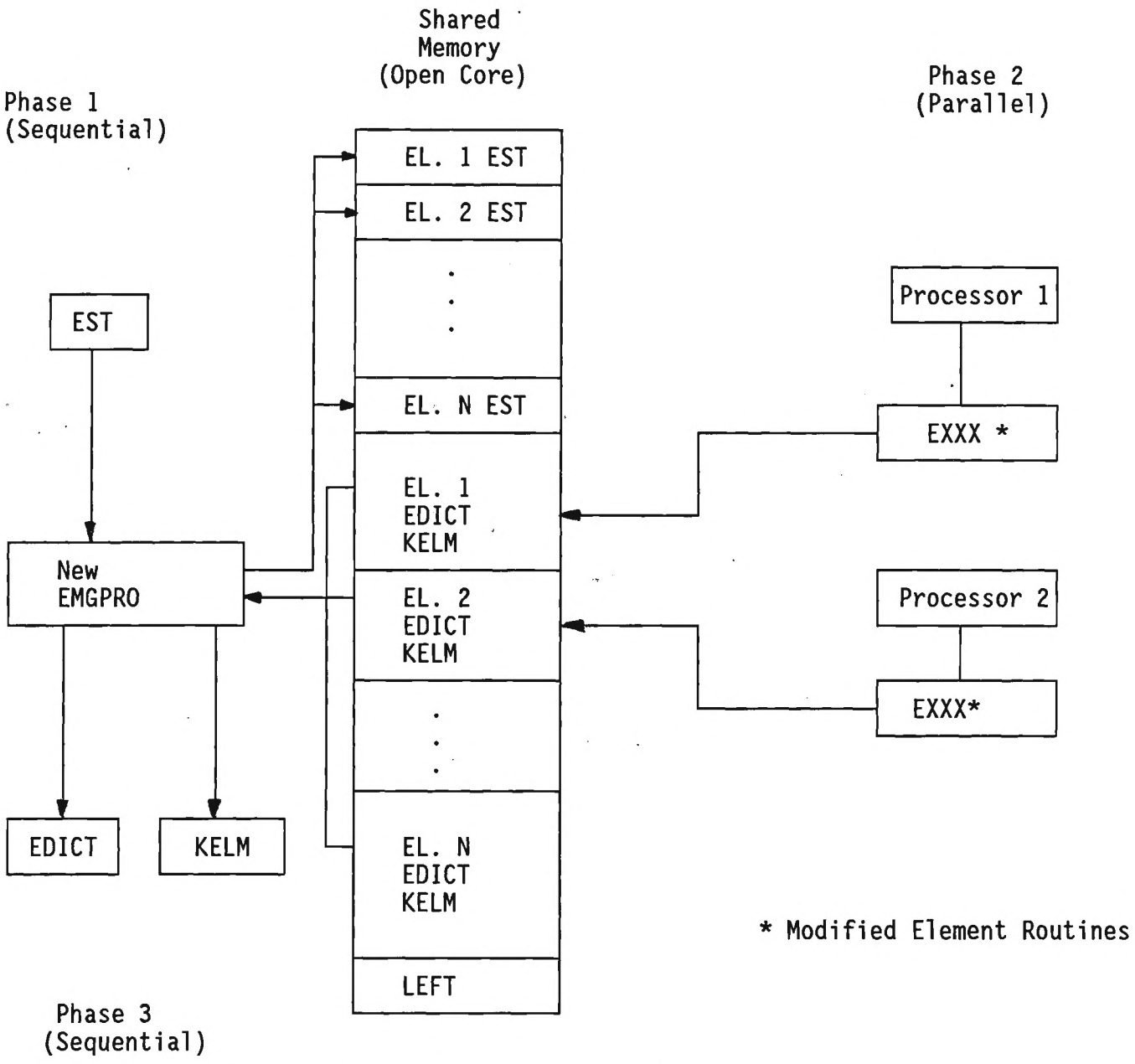


Figure 4.9. Parallel EMG Module Design.

The words per element is the sum of the size of EST, ELDICT and KELM for one element in the currently processed element type. Appropriate pointers are necessary to discriminate element data. ELDICT and KELM of course do not yet exist but appropriate memory is reserved. In Phase 2 EMGPRO spins off parallel tasks mapping the memory resident elements across processors. This could be done in a self-scheduling loop to avoid load balancing problems. In Phase 3 the parallel processes are joined and EMGPRO writes out ELDICT and

Above procedure is particularly attractive because the same approach could be used for parallel and vector processing (compare the proposed multiple element vectorization of the EMG in Section III). The EMGPRO routine would have to be rewritten but only minor changes are required on the element routine level. Changes in the element routines would effect the storage of EST data (currently in a named common block) and the calls to EMGPOM but the vast majority of the element code (literally thousands of statements) would remain unchanged. Absolutely no changes would occur on the GINO level and for the data base.

If parallel GINO routines are available, the I/O could be integrated into the element routines but the open core structure of Figure 4.9 still applies. The number of elements in memory, however, may be limited to the number of processes and each processor uses an assigned workspace.

4.3.2. Parallel EMA in MSC/NASTRAN

The EMA module in MSC/NASTRAN performs basically two tasks: 1.) element stiffness matrices are transformed from local to global coordinates. 2.) the global stiffness matrix is assembled in a columnwise scheme. Several tables (SIL, GPECT, BGPDT, CSTM) are needed and the open core memory is partitioned into two dynamic work areas, one for element stiffness matrices and the other for matrix columns. The memory allocation is such that an optimum mix of global matrix columns and element stiffness matrices is achieved (See MSC/NASTRAN Programmer's Manual).

The basic design of the EMA modul could be used effectively to incorporate parallel processing. The proposed parallel EMA modul would comprise four distinct phases:

Phase 1 - Main process only. Read tabels, memory allocation as described above. Read element stiffness matrices.

Phase 2 - Parallel processes. Coordinate transformation of element stiffness matrices. (Figure 4.10) Elements are mapped across processes in a self scheduling loop, hence load balancing is assured as long the number of elements in memory is considerably larger than the number of processors.

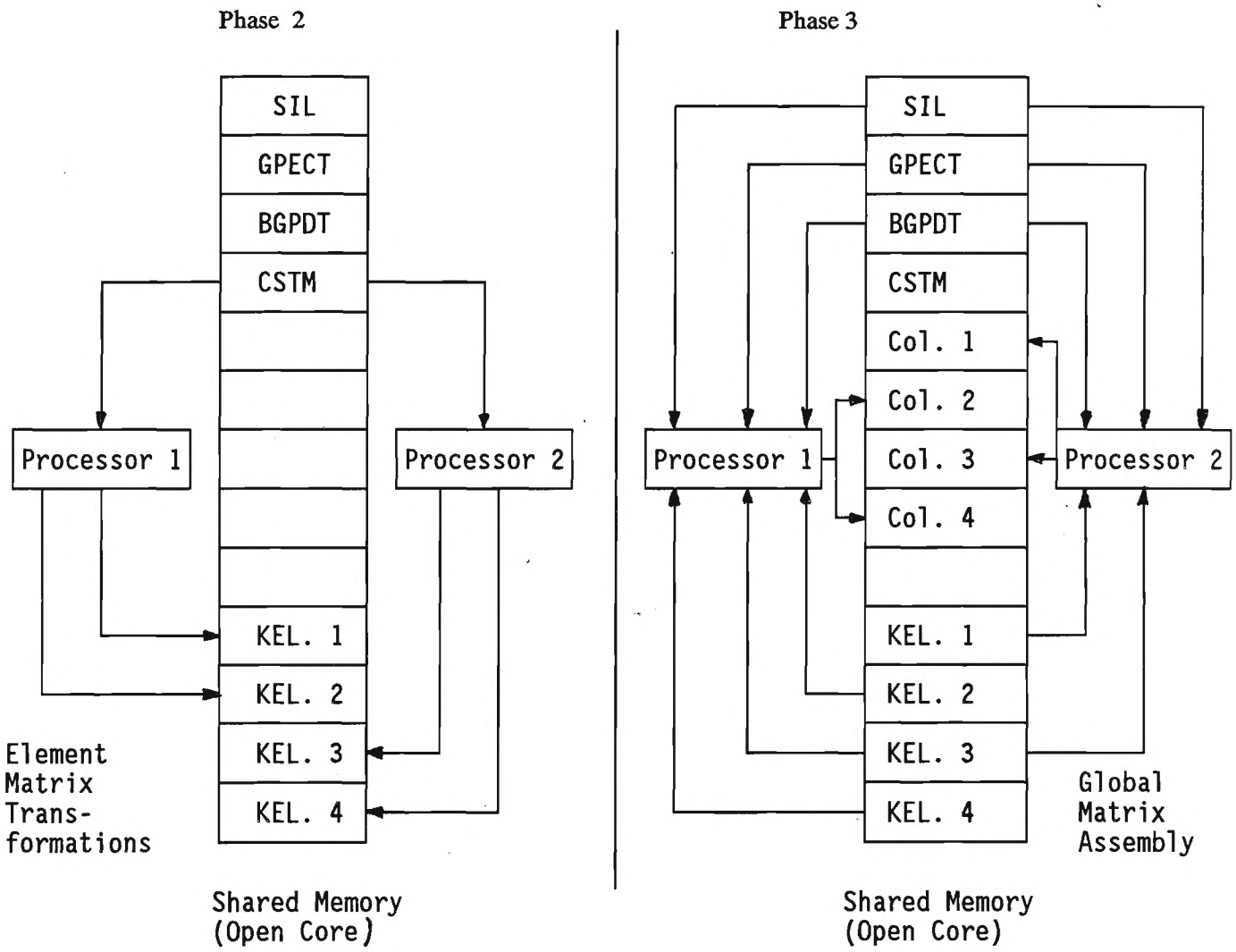


Figure 4.10. Parallel EMA Module Design

Phase 3 - Parallel processes. Each processors claims and assembles complete matrix grid point columns in a self scheduling loop (Figure 4.10). By mapping columns instead of elements across parallel processes, interprocess synchronization can be avoided because there are no critical regions. Load balancing is assured if there are considerably more columns in memory than processors.

Phase 4 - Main process only. Complete matrix columns are written to secondary storage and the procedure can start anew on a different partition of the global stiffness matrix.

Above algorithm is compatible with the current MSC/NASTRAN data base. I/O and table setup are executed sequentially. This limits the achievable parallelism and the significance of these tasks should be measured precisely!

It is interesting to note that as in the proposed parallel EMG phase the same design can be used for multi or vector processing. In Phase 2 vectorized element matrix transformations could be performed over a whole group of computationally equivalent elements.

4.3.3. Parallel SDR in MSC/NASTRAN

The stress and structural data recovery in MSC/NASTRAN resembles quite closely the EMG module. In terms of parallel processing the same concepts apply and the design of parallel code would follow the same approach outlined in section 4.3.1.

4.4 DESIGN APPROACHES TO PARALLEL FEM SYSTEMS

As there are many different procedures to perform an FEM analysis, a variety of approaches to incorporate parallel processing may be conceived. The most important ones are reviewed briefly in the following. The discussion is limited to linear static stress analyses.

A standard FEM procedure for linear statics is built around a global stiffness matrix. Element matrices are generated and assembled, the resulting linear system of equation sis solved and subsequently the element stresses are obtained. Each of the steps are performed in a separate module. Parallelism can be exploited on the level of each module by mapping it across multiple processors (see above).

Alternatively the substructuring method can be used for parallel processing. The model is partitioned in substructures and each substructure may be processed independently on a different processor. This approach can be implemented with moderate changes to existing code if substructuring capabilities are readily available. Only the processing of the global interface requires special attention. As the model is divided into more and more substructures a considerable overhead may be introduced compared to the global method.

The assembly of element stiffness matrices may be combined with a Gaussian elimination of the system matrix by using a frontal solution algorithm. Multi-processing may be incorporated by having multiple fronts running in parallel on different parts of the structure. This approach is somewhat similar to the substructuring method and again special care is required in the interface region where multiple fronts come together.

Element by element conjugate gradient solution schemes are another class of methods which may be used for parallel processing on both vector and multi-processors. The conjugate gradient algorithm can be entirely expressed in terms of matrix-vector multiplication on the element level. A global stiffness matrix does not need to exist and the problem domain may be mapped across multiple processors. Using an appropriate preconditioning scheme this iterative method may become competitive with direct solution techniques for large scale problems. The preconditioning may also be performed on the element level. All but the method of parallel modules require a partitioning of the FEM model in such a way that the computational load of the processors is balanced. Considering a three dimensional model with thousands of elements (Beams, Shells, Solids) and a complex geometry, domain splitting is not trivial and may require significant pre-processing (sequential overhead!).

4.5 CONCLUSIONS AND RECOMMENDED FUTURE WORK

With Version 66 MSC/NASTRAN may be denoted as a moderately parallel software system for large scale linear static solutions. Higher levels of parallelism are desireable if more than four processors are to be employed. A largely parallel (90%) static solution can be conceived by implementing element matrix generation and assembly and the data recovery in parallel. First program designs for parallel EMG and EMA modules are presented here. Parallel code design and data base considerations have to be seen as an integral unit if element dependent data are processed in parallel. [2] The file handling in a parallel environment has been investigated. Future work is necessary to investigate appropriate program designs for nonlinear analysis.

---

2. Parallel code designs for EMG and EMA operations cannot be isolated from the data base.

4.6 REFERENCES

4.1 R. W. Hockney, C. R. Jesshope, "Parallel Computers", Adam Hilger, Bristol, 1981.

4.2 J. A. Fisher, "VLIW Architectures: Supercomputing Via Overlapped Execution", Proceedings of the Second International Conference on Supercomputing, Vol. 2, Santa Clara, CA., May 1987, pp. 353-361.

4.3 M. J. Flynn, "Very High Speed Computing Systems", Proc. IEEE, Vol. 14, 1966, pp. 1901-1909.

4.4 C. L. Seitz, "The Cosmic Cube", Comm. ACM 28, Vol. 1, 1985, pp. 22-23.

4.5 S. Chen, "Large-Scale and High-Speed Multiprocessor System for Scientific Applications: CRAY X-MP Series", NATO ASI Series, Vol. F7, High-Speed Computation, edited by S. J. Kowalik, Springer Verlag Berlin Heidelberg, 1984.

4.6 G. R. Montry, R. E. Benner, "Parallel Processing on a ELXSI 6400", Proceedings of the Second International Conference on Supercomputing, Vol. 2, Santa Clara, CA, May 1987, pp. 64-70.

4.7 G. E. Schmidt, "The Butterfly Parallel Processor", Proceedings of the Second International Conference on Supercomputing, Vol. 1, Santa Clara, CA., May 1987, pp. 362-365.

4.8 N. Matelan, "The Flex/32 Multicomputing Environment in Research in Structures and Dynamics - 1984", NASA CP 2335, 1984.

4.9 O. O. Storaasli, S. W. Peebles, T. W. Crockett, J. D. Knott, L. Adams, "The Finite Element Machine: An Experiment in Parallel Processing:, NASA TM 84514, 1982.

4.10 D. H. Norrie, C. W. Norrie, "Architecture and Program Structures for a Special Purpose Finite Element Computer", EDF-Bull. Direct. Etud. Rech. C., Math., Informat. 1, 1983, pp. 103-108.

4.11 J. J. Dongarra, F. G. Gustavson, A. Karp, "Implementing Linear Algebra Algorithms on a Vector Pipeline Machine", SIAM Review, 26, 1984, pp. 91-112.

4.12 P. M. Kogge, "Algorithm Development for Pipelined Processors", Proceedings 1977 International Conference Parallel Processing, IEEE No. 77, CH1253-4C, August 1977.

4.13 D. Kuck, D. Lawre, A. Samek, "High Speed Computer and Algorithm Organization", Academic Press, 1977.

4.14 J. J. Dongarra, C.D. C. Sorensen, "Linear Algebra on High-Performance Computers", Proceedings of the Second International Conference on Parallel Computing Held at the Freie Universitaet Berline, September 1985, North Holland, Amsterdam, 1986, pp. 3-23.

4.15 G. L. Goudreau et al., "Efficient Large-Scale Finite Element Computations in a CRAY Environment", Impact of New Computing Systems on Computational Mechanics (edited by A. K. Noor), ASME, 1983, pp. 141-154.

4.16 V. B. Venkayya et al., "Structural Optimization on Vector Processors", Impact of New Computing Systems on Computational Mechanics (edited by A. K. Noor), ASME, 1983, pp. 155-170.

4.17 G. M. Baudet, "Asynchronous Iterative Methods for Multiprocessors", Journal Assoc. Comput. Mach., Vol. 25, 1978, pp. 226-244.

4.18 V. Conrad, Y. Wallach, "Iterative Solution of Linear EQuations on a Parallel Processor System", IEEE Trans. Comput., Vol. 26, 1977, pp. 838-847.

4.19 L. M. Adams, "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers", NASA CR-166027, NASA Langley Research Center, Hampton, VA, 1982.

4.20 J. S. Kowalik, S. P. Kumar, "An Efficient Parallel Block Conjugate Gradient Method for Linear Equations", Proceedings of the 1982 International Conference on Parallel Processing, IEEE 82CH1794-7, 1982, pp. 47-52.

4.21 M. L. Patrick, T. W. Pratt, "Communications Oriented Programming of Parallel Iterative Solutions of Sparse Linear Systems", Communications in Applied Numerical Methods, Vol. 2, 1986, pp. 255-261.

4.22 L. M. Adams, J. M. Ortega, "A Multicolor SOR Method for Parallel Computaton", Proceedings of the 1982 International Conference on Parallel Processing, IEEE 82CH1794-7, 1982, pp. 53-56.

4.23 S. -L. Sheu, W. Lin, C. R. Das, "An Efficient Parallel Algorithm of Conjugate Gradient Methods", Proceedings of the Second International Conference on Supercomputing, Vol. 2, Santa Clara, CA., May 1987, pp. 488-496.

4.24 S. L. Scott, "Comparison of Parallel SOR Algorithms for Solution of Sparse Matrix Problems", Proceedings of the Second International Conference on Supercomputing, Vol. 1, Santa Clara, CA., May 1987, pp. 424-432.

4.25 M. Dubois, "Performance of SOR Algorithms in Multiprocessors", Proceedings of the SEcond International Conference on Supercomputing, Vol. 2, Santa Clara, CA, May 1987, pp. 414-423.

4.26 A. Hayashi, S. Utku, R. J. Melosh, "Variation in Efficiency of Parallel Algorithms", Computers and Structures, Vol. 21, No. 5, 1985, pp. 1025-1034.

4.27 A. George, M. T. Heath, J. Liu, "Parallel Cholesky Factorization on a Shared-Memory Multiprocessor", Tech. Rept. ORNL-6124, Oak Ridge National Laboratory, March 1985.

4.28 M. T. Heath, "Parallel Cholesky Factorization in a Message-Passing Multiprocessor Environment", Tech. Retp. ORNL-6150, Oak Ridge National Laboratory, March 1985.

4.29 S. Utku, M. Salamana, R. J. Melosh, "Concurrent Cholesky Factorization of Positve Definite Banded Hermitian Matrices", International Journal of Numerical Methods in Engineering, Vol. 23, 1986, pp. 2137-2175.

4.30 R. J. Melosh, S. Utku, M. Salamana, "Direct Finite Element Equation Solving Algorithms", Computers and Structures, Vol. 20, No. 1-3, 1985, pp. 99-105.

4.31 G. A. Geist, M. T. Heath, J. Liu, "Parallel Cholesky Factorization on a Hypercube Multiprocessor", Tech. Rept. ORNL-6190, Oak Ridge National Laboratory, July 1985.

4.32 E. O'Neil, H. Allik, S. Moore, "Finite Element Analysis on the BBN Multiprocessor", Proceedings of the Second International Conference on Supercomputing, Vol. 1, Santa Clara, CA., May 1987, pp. 366-374.

4.33 S. S. Chen, J. J. Dongarra, "Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences with Small Granularity", Journal of Parallel and Distributed Computing, 1, 1984, pp. 22-31.

4.34 L. Komzsik, "Parallel Decomposition Technique on a Double Level Storage System", International Conference for Vector and Parallel Processing, Leon, Norway, 1986.

4.35 L. Komzsik, "Parallel Static Solution in Finite Element Analysis", Proceedings of the Second International Conference on Supercomputing, Vol. 2, Santa Clara, CA, May 1987, pp. 138-144.

4.36 D. Goehlich, L. Komzsik, R. E. Fulton, "Decomposition of Finite Element Matrices on Parallel Computers", Proceedings of the 1987 ASME International Computers in Engineering Conference, Vol. 3, New York, NY, August 1987, pp. 415-521.

4.37 R. G. Grimes, H. D. Simon, "Dynamic Analysis with the Lanczos Algorithm", Proceedings of the Second International Conference on Supercomputing, Vol. 2 Santa Clara, CA, May 1987, pp. 110-118.

4.38 S. Bostic, R. E. Fulton, "A Concurrent Processing Approach to Structural Vibration Analysis", 26th AIAA/ASME/ASCE/AHA Structures, Structural Dynamics and Materials Conference, Orlando, FL, AIAA Paper No. 85-078-CP, April 1985.

4.39 S. Bostic, R. E. Fulton, "Experiences with the Lanczos Method on a Parallel Computer", Proceedings of the 1987 ASME International Computers in Engineering Conference, Vol. 2, New York, NY, August 1987, pp. 353-360.

4.40 O. Storaasli, J. R. Ransom, R. E. Fulton, "Structural Dynamic Analysis on a Parallel Computer: The Finite Element Machine", 25th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Palm Springs, CA, May 1984.

4.41 S. Doi, S. Koyama, "A Parallel Computation Technique for Finite Element Methods", Systems - Computers - Controls, Vol. 13, No. 2, 1982, pp. 76-84.

4.42 A. K. Noor, O. O. Storaasli, R. E. Fulton, "Finite Element Technology of the Future", Impact of New Computers on Computational Mechanics, ASME Special Publication H00275, November 1983, pp. 1-32.

4.43 G. F. Carey, "Parallelism in Finite Element Modelling", Communications in Applied Numerical Methods, Vol. 2, 1986, pp. 281-287.

4.44 F. Ercal et al., "Parallel Computers for Finite Element Analysis", Proceedings of the 1986 ASME International Computers in Engineering Conference, Vol. 2, Chicago, Il., July 1986, pp. 43-50.

4.45 C. P. Blankenship, R. J. Hayduk, "Potential Supercomputer Needs for Structural Analysis", Proceedings of the SEcond International Conference on Supercomputing, Vol. 2, Santa Clara, CA, May 1987, pp. 180-202.

4.46 K. H. Law, "A Parallel Finite Element Solution Method", Computers and Structures, Vol. 23, No. 6, 1986, pp. 845-858.

4.47 GF. A. Lyzenga, A. Raefsky, G. H. Hager, "Finite Elements and the Method of Conjugate Gradients on a Concurrent Processor", Proceedings of the 1985 ASME International Computers in Engineering Conference, Vol. 3, Boston, MA., August 1985, pp. 401-405.

4.48 B. Nour-Omid, K. C. Park, "Solving Structural Mechanics Problems on the Caltech Hypercube Machine", Computer Methods in Applied Mechanics and Engineering, Vol. 61, 1987, pp. 161-176.

4.49 H. Allik, W. Crowther, J. Goodhue, S. Moore, R. Thomas, "Implementation of the Finite Element Method on the Butterfly Parallel Processor", Proceedings of the 1985 ASME international Computers in Engineering Conference, Vol. 3, Boston, MA, August 1985, pp. 393-399.

4.50 H. Allik, S. Moore, "Finite Element Software on a Multiprocessor", Proceedings of the Ninth Conference on Electronic Computation, ASCE, Birmingham, AL., February 1986, pp. 680-691.

4.51 C. Farhat, E. Wilson, "A New Finite Element Concurrent Computer Program Architecture", International Journal for Numerical Methods in Engineering, Vol. 24, 1987, pp. 1771-1792.

4.52 C. Farhat, "Multiprocessors in Computational Mechanics", Ph.D. Dissertation, University of California, Berkeley, CA., 1986.

4.53 O. O. Storaasli, P. Bergan, "Nonlinear Substructuring Method for Concurrent Processing Computers", AIAA Journal, Vol. 25, No. 6, June 1987, pp. 871-876

4.54 R. E. Fulton, "The Finite Element Machine: An Assessment of the Impact of Parallel Computing on the Future of Finite Element Computations", Finite Elements in Analysis and Design, North Holland, pp. 83-96.

SECTION V

PARALLEL NUMERICAL INTEGRATION METHODS FOR NONLINEAR DYNAMICS

5.1 INTRODUCTION

This section reports on the parallel implementation of the central difference, Newmark, Houbolt and Wilson theta methods for nonlinear dynamic solutions. Comparisons between the explicit central difference method and the implicit methods are made from the view point of parallel computation. A comparison is also made between the central difference method incorporating the Cholesky decomposition method and the central difference method incorporating a mixed Jacobi/Gauss-Seidel method.

Besides the commonly used integration methods, two predictor-corrector methods - the Adams-Molton method and the Kunz method are also implemented in parallel. Finally, the implementation of the parallel central difference and Newmark methods for solving a plane stress problem is described and comparison between the two methods is made.

5.2 FORMULATION OF PROBLEM

In general the equilibrium equation in nonlinear finite element dynamic analysis can be expressed in the form

$$M\ddot{u} + C\dot{u} + F(u,t) = f(u,t) \tag{5.1}$$

where

u - displacement vector

$\dot{u}$ - velocity vector

$\ddot{u}$ - acceleration vector

M - mass matrix

C - damping matrix

F - nodal point force vector corresponding to the element stresses

To solve Eq. (5.1) for the response one can use explicit methods such as the central difference method, or implicit methods such as the Newmark, Houbolt and Wilson theta methods.

If an implicit integration method is used, some kind of equilibrium iterations should be performed in the step-by-step incremental analysis because the solution of the displacements at $t + \Delta t$ is based on using the equilibrium condition at time $t + \Delta t$. A widely used equilibrium iteration scheme is the modified Newton iteration. Applying this iteration in an implicit integration method yields

$$M\ddot{u}^{(i)}_{t+\Delta t} + C\dot{u}^{(i)}_{t+\Delta t} + K_t \Delta u^{(i)} = f_{t+\Delta t} - F^{(i-1)}_{t+\Delta t}$$

$$u^{(i)}_{t+\Delta t} = u^{(i-1)}_{t+\Delta t} + \Delta u^{(i)}$$

with the initial conditions

$$u^{(o)}_{t+\Delta t} = u_t$$

$$F^{(o)}_{t+\Delta t} = F_t$$

where $i = 1, 2, 3...$ are the iteration index and K_t is the tangent stiffness matrix which corresponds to the geometric and material condition at time t. Other iteration schemes could of course be used such as the BFGS method or the line search method.

If an explicit integration method is used, equilibrium iterations are not required since the solution of the displacements at $t + \Delta t$ is based on using the equilibrium condition at time t.

5.3 A SIMPLE TEST PROBLEM

The transient response problem shown in Figure 5.1 is one of the test problems used in the study. The problem is a discrete mass system with nonlinear behavior. In Figure 5.1, S_{n1}, S_{n2}, ..., S_{nn} are nonlinear springs where the spring force is proportional to the cubic of the displacements.

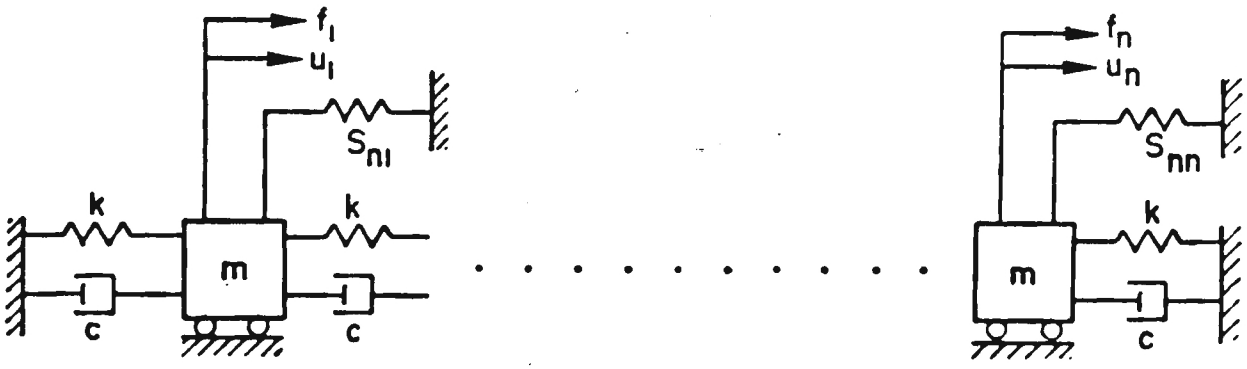


Figure 5.1. A Sample Problem.

For such a simple case, Equation (5.1) can be expressed in the following form:

$$M\ddot{u} + C\dot{u} + Ku = f \tag{5.2}$$

where K is the stiffness matrix of the system. Since the springs are nonlinear, K is not constant and requires updating to current values at every integration step.

To solve Equation (5.2) the explicit central difference method [5.1] is first used in this study. The method is based on the following approximation:

$$\dot{u}_t = \frac{1}{2\Delta t}\left(u_{t+\Delta t} - u_{t-\Delta t}\right) \tag{5.3}$$

$$\ddot{u}_t = \frac{1}{\Delta t^2}\left(u_{t+\Delta t} - 2u_t + u_{t-\Delta t}\right) \tag{5.4}$$

Substituting (5.3) and (5.4) into (5.2) gives

$$\bar{M}u_{t+\Delta t} = \bar{f}_t \tag{5.5}$$

where the effective mass matrix $\bar{M}$ and effective force vector $\bar{f}_t$ are

$$\bar{M} = \frac{1}{\Delta t^2} M + \frac{1}{2\Delta t} C \tag{5.6}$$

$$\bar{f}_t = f_t - \left[K - \frac{2}{\Delta t^2} M\right] u_t - \left[\frac{1}{\Delta t^2} M - \frac{1}{2\Delta t} C\right] u_{t-\Delta t} \tag{5.7}$$

The implicit methods such as the Newmark [5.2], Houbolt [5.3] and Wilson theta [5.4] methods are also used in this study to solve Equation 5.2. As the first stage of study, the equilibrium iteration procedure was not included in solving the simple test problem.

The approximation used in the Newmark methods is

$$\dot{u}_{t+\Delta t} = \frac{\alpha}{\beta\Delta t}\left[u_{t+\Delta t} - u_t\right] - \left[\frac{\alpha}{\beta} - 1\right]\dot{u}_t - \Delta t\left[\frac{\alpha}{2\beta} - 1\right]\ddot{u}_t \tag{5.8}$$

$$\ddot{u}_{t+\Delta t} = \frac{1}{\beta \Delta t^2}\left[u_{t+\Delta t} - u_t\right] - \frac{1}{\beta \Delta t}\dot{u}_t - \left[\frac{1}{2\beta} - 1\right]\ddot{u}_t \tag{5.9}$$

Substituting Equation (5.8) and(5.9) into (5.2) yields

$$\bar{M}u_{t+\Delta t} = \bar{f}_{t+\Delta t} \tag{5.10}$$

where

$$\bar{M} = \frac{1}{\beta \Delta t^2} M + \frac{\alpha}{\beta \Delta t} C + K \tag{5.11}$$

$$\bar{f}_{t+\Delta t} = f_{t+\Delta t} + \left[\left(\frac{1}{2\beta} - 1\right)M + \Delta t\left(\frac{\alpha}{2\beta} - 1\right)C\right]\ddot{u}_t + \left[\frac{1}{\beta \Delta t} M + \left(\frac{\alpha}{\beta} - 1\right)C\right]\dot{u}_t \tag{5.12}$$

The approximation used in the Houbolt method is

$$\dot{u}_{t+\Delta t} = \frac{1}{6\Delta t}\left[11u_{t+\Delta t} - 18u_t + 9u_{t-\Delta t} - 2u_{t-2\Delta t}\right] \tag{5.13}$$

$$\ddot{u}_{t+\Delta t} = \frac{1}{\Delta t^2}\left[2u_{t+\Delta t} - 5u_t + 4u_{t-\Delta t} - u_{t-2\Delta t}\right] \tag{5.14}$$

Substituting (5.13) and (5.14) into (5.2) gives

$$\bar{M}u_{t+\Delta t} = \bar{f}_{t+\Delta t}$$

where

$$\bar{M} = \frac{2}{\Delta t^2} M + \frac{11}{6\Delta t} C + K$$

$$\bar{f}_{t+\Delta t} = f_{t+\Delta t} + \left[\frac{5}{\Delta t^2} M + \frac{3}{\Delta t} C\right] u_t - \left[\frac{4}{\Delta t^2} M + \frac{3}{2\Delta t} C\right] u_{t-\Delta t}$$

$$+ \left[\frac{1}{\Delta t^2} M + \frac{1}{3\Delta t} C\right] u_{t-2\Delta t}$$

The Wilson theta method [5.4] is also used in the study. This method is based on the following difference formulas:

$$\dot{u}_{t+\theta\Delta t} = \frac{3}{\theta\Delta t} [u_{t+\theta\Delta t} - u_t] - 2\dot{u}_t - \frac{\theta\Delta t}{2} \ddot{u}_t \tag{5.15}$$

$$\ddot{u}_{t+\theta\Delta t} = \frac{6}{\theta^2\Delta t^2} \left[u_{t+\theta\Delta t} - u_t\right] - \frac{6}{\theta\Delta t} \dot{u}_t - 2\ddot{u}_t \tag{5.16}$$

Substituting (5.15) and (5.16) into (5.2) yields

$$\bar{M} u_{t+\theta\Delta t} = \bar{f}_{t+\theta\Delta t}$$

where

$$\bar{M} = \frac{6}{\theta^2\Delta t^2} M + \frac{3}{\theta\Delta t} C + K$$

$$\bar{f}_{t+\theta\Delta t} = f_{t+\theta\Delta t} + \left[\frac{6}{\theta^2\Delta t^2} M + \frac{3}{\theta\Delta t} C\right] u_t$$

$$+ \left[\frac{6}{\theta\Delta t} M + 2C\right] \dot{u}_t + \left[2M + \frac{\theta\Delta t}{2} C\right] \ddot{u}_t$$

5.4 OVERVIEW AND COMPARISONS OF PARALLEL INTEGRATION METHODS

Parallel numerical algorithms are typically extensions of sequential numerical algorithms. Therefore, when parallel algorithms are evaluated, one should start from the criteria that are appropriate for the evaluation of sequential algorithms. In the case of integration procedures, for example, stability, accuracy and computational efficiency should be considered. Stability is an essential criteria for an algorithm. If an algorithm is unstable, any error resulting from the numerical integration or round-off can grow and make response calculations worthless. Accuracy is another important feature of algorithm. It is usually characterized by the order of approximation used in the algorithm. Larger time step size can be used for enough computational accuracy if the order of approximation is higher. Therefore, the accuracy of an algorithm also has some effects on its computational efficiency. Another factor which affects the computational efficiency of an algorithm is the operation count. To fulfill the same task the number of operations needed for different algorithms can be different.

In addition to the intrinsic properties of sequential algorithms, one should also examine those aspects brought by parallel computations, such as parallel computational effectiveness and the effects of parallel computation on stability and accuracy (if such effects exists). Parallel effectiveness is an important concept in parallel computation. Synchronization and communication among processors are required for the implementation of parallel algorithms and those operations need additional execution time. Furthermore the computation task may not be equally distributed to each processor and the workloads of processors can be unbalanced. Moreover, in many cases not each part of an algorithm can be implemented in parallel. The speedup is degraded significantly when the portion of sequential computation is large. All the above factors will affect the parallel effectiveness of an algorithm.

To improve the parallel effectiveness of an algorithm the synchronization, communication and the sequential part of computation should be minimized and the workload distribution on processors should be as even as possible. And one of the purposes of this research is to identify those methods which have better parallel effectiveness and to modify the present algorithms to improve their parallel effectiveness.

In the following part of the report the central difference method and the Newmark, Houbolt, Wilson-theta methods are investigated from the view point of parallel computation. They are chosen because they are popular in finite element codes and are promising methods in parallel computation (at least in some cases). For example, when the central difference method is used in the case of diagonal mass matrix, the equations are decoupled and therefore the communication is minimized. Furthermore, the algorithm is simple to formulate and implement in such a case. The Newmark, Houbolt and Wilson-theta methods are also attractive because they are unconditionally stable. Although equation solving is involved, the parallel implementation of decomposition is feasible.

The four methods are investigated and compared more specifically in the following.

The central difference method is an explicit method. It is conditionally stable, which means that one should not select an integration time step larger than a critical value; otherwise the computed response may become unstable. The stable criteria are given in references such as [5.1]. The Newmark, Houbolt and Wilson-theta methods are implicit methods. They are unconditionally stable (for some range of integration parameters in the case of Newmark and Wilson-theta), which means that one can choose an integration time step without concern for computational instability. As for accuracy and computational efficiency, all four algorithms have errors of order $O(\Delta t^2)$ and in general are efficient in sequential computation. It should be noticed that when the central difference method is used in the case of a diagonal mass matrix and a diagonal (or neglected) damping matrix, the effective mass matrix $\bar{M}$ in Equation (5.5) is diagonal. Therefore, the solution at the next time step can be obtained directly and the computational efficiency is high. In the case where the mass matrix and/or damping matrix are nondiagonal, however, Equation (5.5) becomes a coupled set of equations and the central difference method requires the solution of sets of equations. The same requirements exists if the implicit methods are used. Therefore, solving sets of simultaneous equations can be a significant part of the nonlinear dynamic solution.

As for parallel effectiveness the central difference method has very good parallelism when it is used in the case of a diagonal mass matrix and a diagonal (or neglected) damping matrix since the equations are decoupled and no equation solving is involved. In contrast, if the implicit methods are used in this case, the parallel effectiveness will be effected by equation solving.

To compare the explicit central difference method with the implicit methods in the case of diagonal mass matrix, some numerical experiments are performed to solve the test problem shown in Figure 5.1. In the parallel central difference method the parallelism is exploited simply by assigning some of the equations in (5.6), (5.7) and (5.5) to each processor and the parallelism is high. Some synchronizations are needed in performing the calculations. For example, for one step to another step, synchronization is required. Figure 5.2 is the flowchart for the implementation of the method. In the implicit methods the situation is more complicated because the solution of the simultaneous equation is involved at each time step.

Reference [5.6] shows that for tridiagonal systems the parallel cyclic reduction method has a much higher computational efficiency and parallel effectiveness than the parallel Cholesky decomposition method. Therefore, the parallel cyclic reduction method is selected to be incorporated in the implicit methods in the numerical experiments. The cyclic reduction method [5.6-5.8] is an elimination method for tridiagonal type equations where the reduction strategy is based on eliminating every other variable and equation. These elimination steps are decoupled and can be assigned to independent processors to exploit parallelism. Since equation solving is involved in the implicit methods, the operation counts at each time step are increased and more synchronizations are needed. The flowchart for the parallel Newmark method incorporating parallel cyclic reduction is shown in Figure 5.3.

The comparative computation results for the explicit central difference method and the implicit methods incorporating cyclic reduction are shown in Figure 5.4 through Figure 5.6. The relative speedup in Figures 5.4-5.6 is defined as the execution time of the central difference method on one processor divided by the execution time of

the implicit method examined on n processors. Therefore, if the relative speedup of an implicit method is less than that of the central difference method for a specific number of processors, that will mean that the computational speed of that implicit method is slower than that of the central difference method for that specific number of processor. Figure 5.4 through Figure 5.6 show that the computational efficiency and parallel effectiveness are similar for the three implicit methods. The figures also show that the computational efficiency and parallel effectiveness of the central difference method is much higher than the implicit methods in the case of diagonal mass matrix and diagonal (or neglected) damping matrix. Therefore, the parallel explicit central difference method is recommended in such a case.

On the other hand, if the mass matrix and/or damping matrix are nondiagonal, the central difference method will also require solution to simultaneous equations. Its advantages existing in the case of diagonal mass matrix will therefore disappear. This point will be verified by the computation results obtained in solving a plane stress problem in parallel which is to be mentioned in Section 5.7. Therefore, the central difference method is not recommended for use in this case generally because of its conditional stability. But in some special cases, the central difference method still has some advantage. For example, in the case of singular stiffness matrix, the implicit method is susceptible to diverge, whereas the explicit methods usually behaves acceptably. Such cases can happen if the material fails by cracking or plastic buckling (e.g. in crash dynamics).

To achieve higher parallel efficiency and associated performance improvement one should consider limiting the models to a diagonal mass matrix. Any loss of accuracy due to lumped mass can be compensated by use of finer finite element discretization.

Table 5.1 summarizes the characteristics of the parallel central difference and the parallel Newmark methods.

CASE	DIAGONAL MASS MATRIX		NONDIAGONAL MASS MATRIX	
Method	Central Difference	Newmark	Central Difference	Newmark
Stability	Conditionally Stable	Unconditionally Stable	Conditionally Stable	Unconditionally Stable
Accuracy	$O(\Delta t^2)$	$O(\Delta t^2)$	$O(\Delta t^2)$	$O(\Delta t^2)$
Computational Efficiency	High	Medium	Medium	Medium
Parallel Effectiveness	High	Medium	Medium	Medium

Table 5.1 A Comparison Between the Parallel Central Difference and the Parallel Newmark Methods.

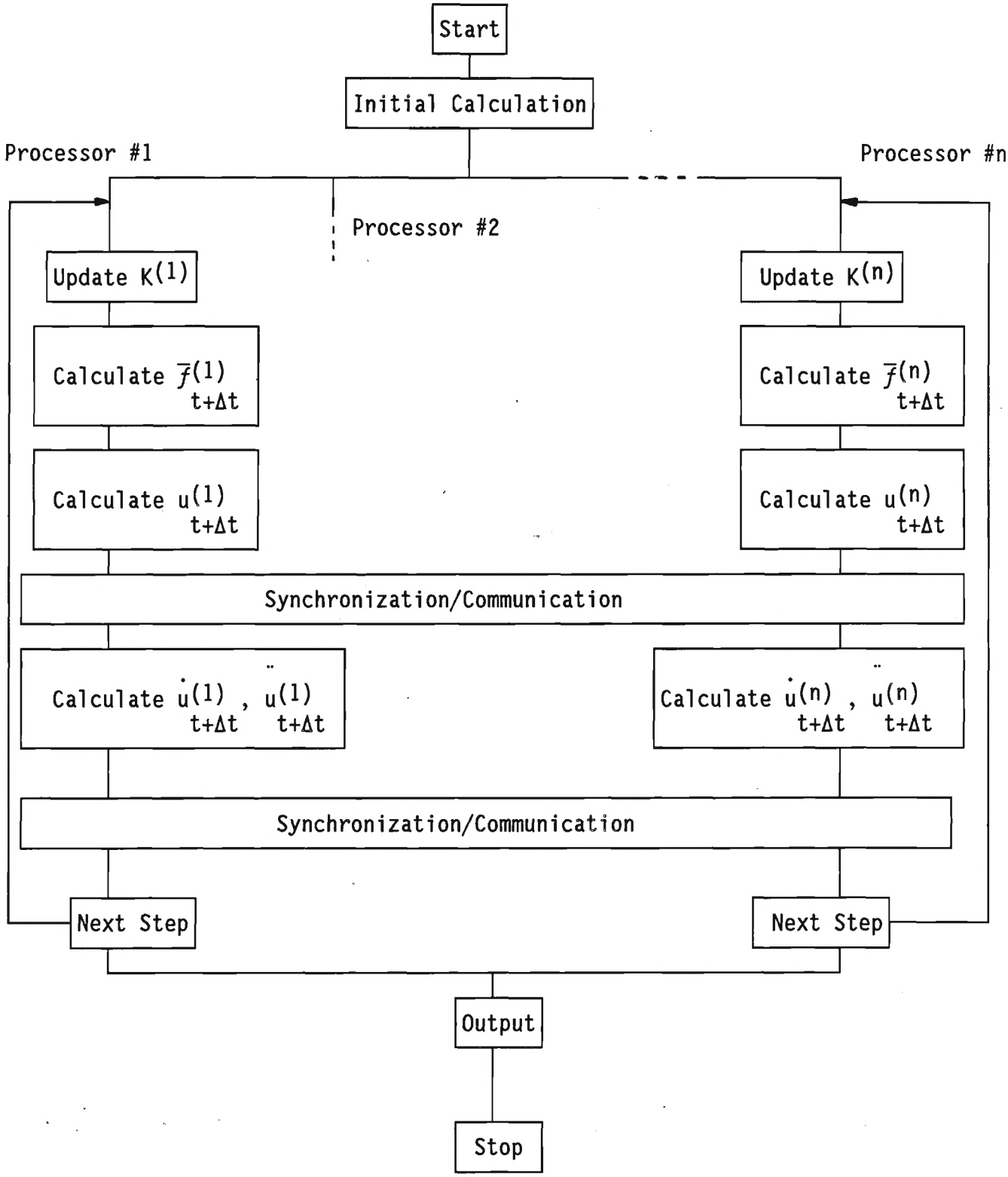


Figure 5.2. The Parallel Central Difference Method (Diagonal Mass Matrix).

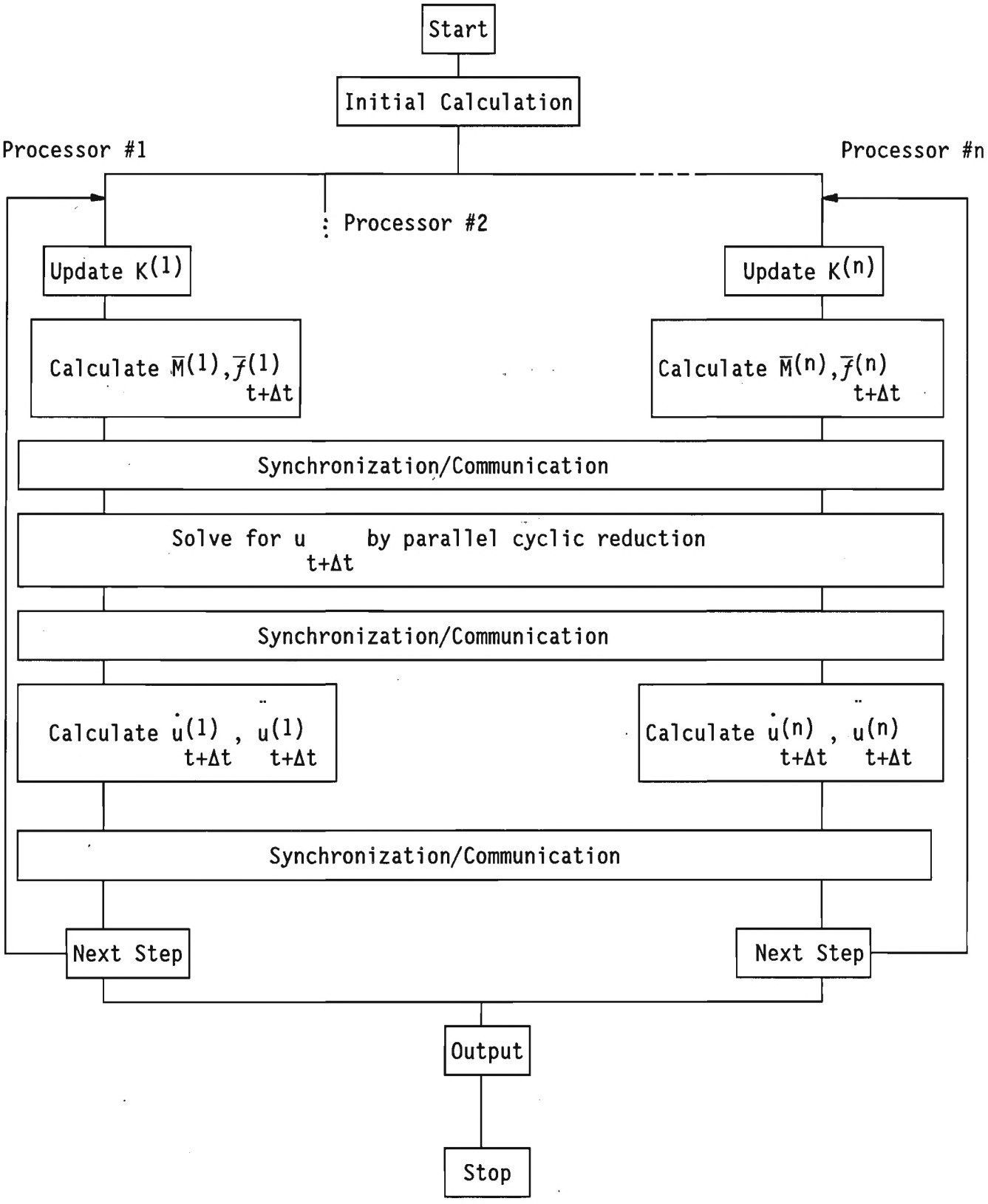


Figure 5.3. The Parallel Newmark Method Incorporating the Parallel Cyclic Reduction.

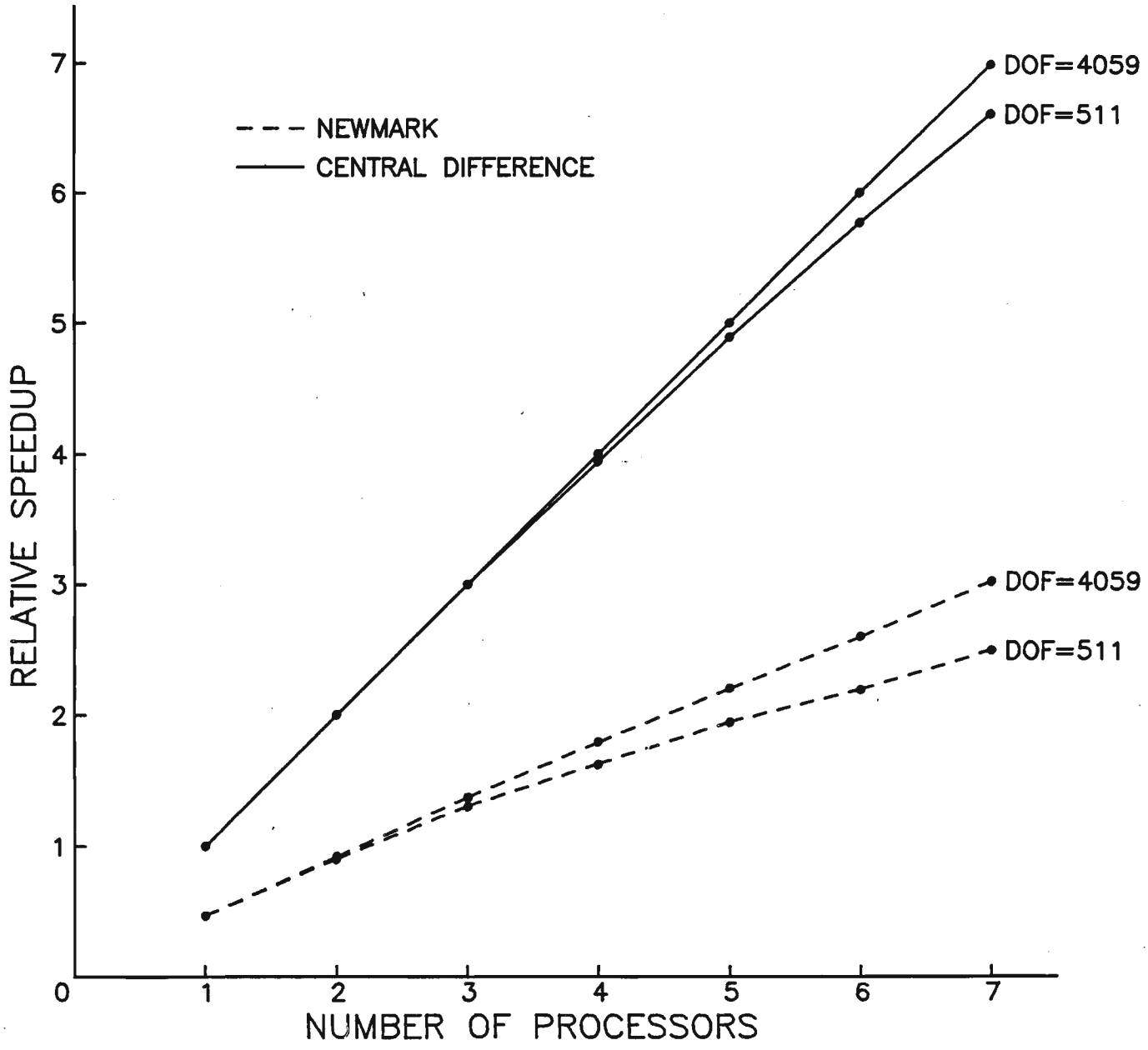


Figure 5.4. The Central Difference Method Versus the Newmark Method.

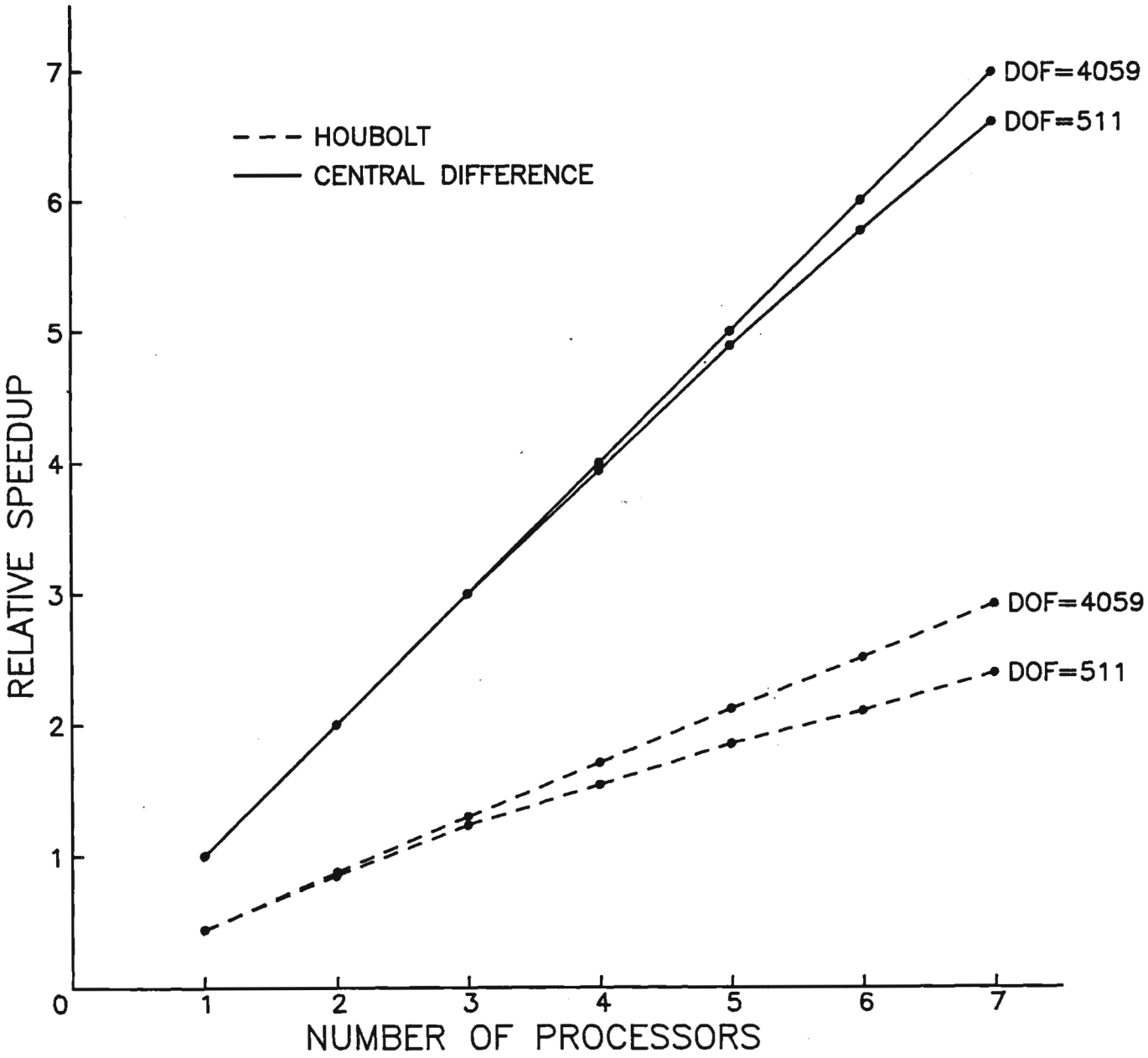


Figure 5.5. The Central Difference Method Versus the Houbolt Method.

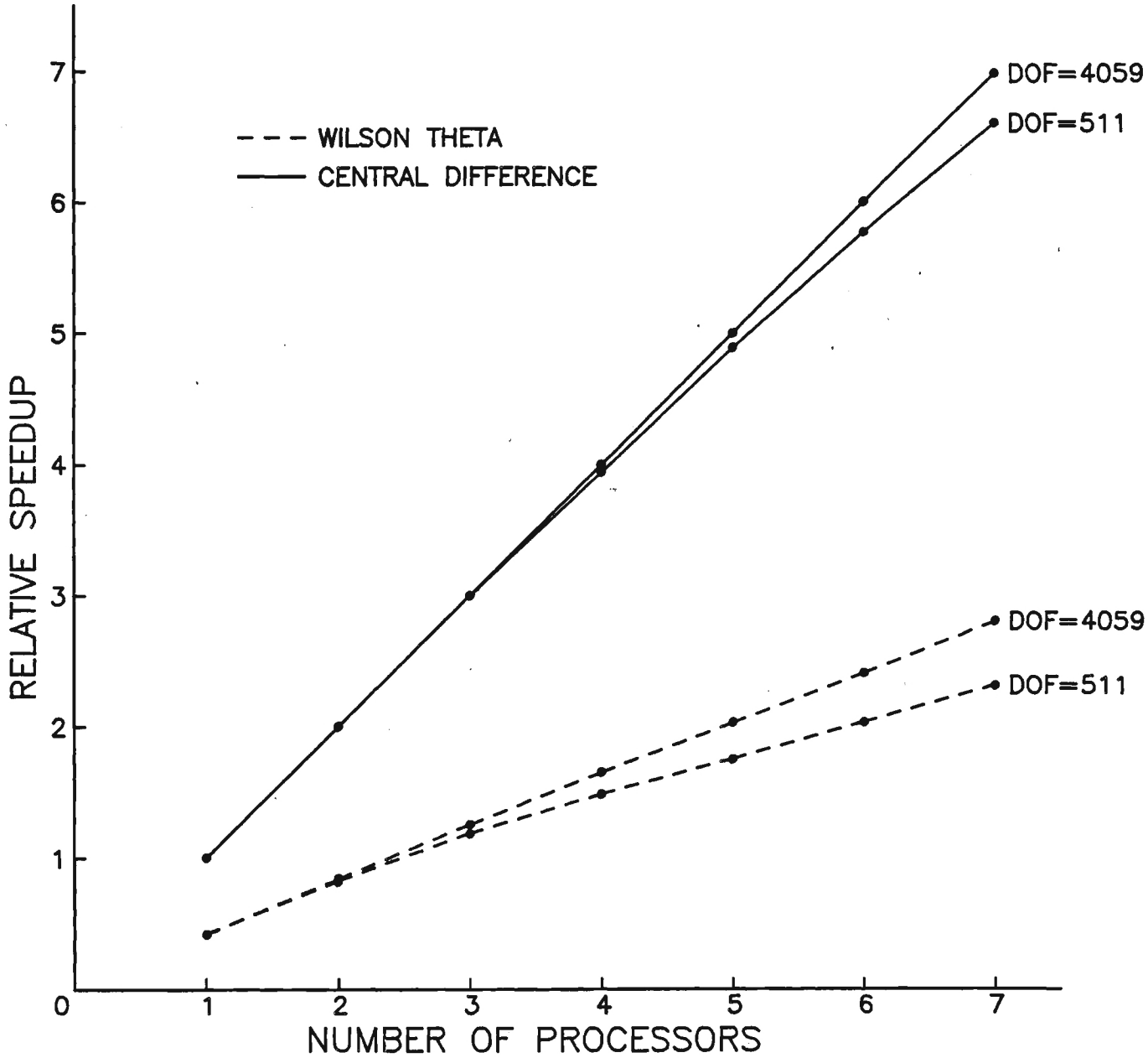


Figure 5.6. The Central Difference Method Versus the Wilson-Theta Method.

5.5 CHOLESKY VERSUS MJGS IN INTEGRATION

As noted above, solving sets of equations can be a significant part of the nonlinear dynamic solution. Therefore, the properties of several parallel equation solving techniques when combined with parallel integration procedures must be studied from the view point of parallel computation. One of the investigations studied herein is a comparison between the parallel implementations of the Cholesky LDL^T decomposition method and a mixed Jacobi/Gauss-Seidel (MJGS) method when incorporated in the parallel central difference method.

The Cholesky decomposition method is a direct method widely used in commercial finite element codes. Its parallel effectiveness is quite high when the bandwidth is large. But if the bandwidth is small, the parallel effectiveness is significantly degraded. The mixed Jacobi/Gauss-Seidel method is an iterative method which has good parallelism. But it may have convergence difficulties in some cases. The two methods represents two different categories of methods and therefore were chosen to be studied.

The numerical experiments are performed on the same test problem mentioned above but the mass matrix is modified in the following way to produce nondiagonal mass matrices:

$$M = \begin{pmatrix} m & 1 & \cdots & 1 & & & \\ 1 & \ddots & \ddots & & & 0 & \\ & \ddots & & \ddots & & & \\ & & \ddots & & & & 1 \\ 1 & \cdots & & & & & \vdots \\ & 0 & & & & & 1 \\ & & \cdots & 1 & \cdots & 1 & m \end{pmatrix}$$

The value of m chosen in the numerical experiments is 50.

References [5.5] and [5.6] show that neither the parallel Cholesky decomposition method nor the cyclic reduction method are efficient when the half bandwidth is of an intermediate range (say, from 5 to 50). Therefore, the half bandwidth for the numerical experiments in this study are chosen as the intermediate range.

The concurrency in parallel Cholesky LDL^T decomposition here is achieved by distributed execution of the factorization and the parallel granularity is realized on the column level [5.5]. Suppose that the number of

processors is p, then each column has p-1 synchronization points. The distribution of columns among processors is done in a self-scheduling loop; i.e., when a processor has finished its column it automatically identifies and claims the next not yet factored column. Figure 5.7 is the flowchart for the parallel central difference method incorporating the parallel Cholesky decomposition method.

The mixed Jacobi/Gauss-Seidel method used here updates values according to the computations within each processor but does not update values between processors. Therefore, the method is Gauss-Seidel within each processor and Jacobi between processors. For small numbers of processors it has the efficiency of the Gauss-Seidel while retaining the parallel benefits of Jacobi. Figure 5.8 shows the logic of the method and processor assignment, and Figure 5.9 is the flowchart for the parallel implementation of the method. The flowchart of the parallel central difference method incorporating the parallel MJGS method is shown in Figure 5.10.

The comparative results are shown in Figure 5.11 through Figure 5.14. Figure 5.11 and Figure 5.12 are for the case where 100 time step integration is performed. The results of the case where a 5 time step integration is performed are shown in Figure 5.13 and Figure 5.14. For convenience two abbreviations will be used in the rest of this report; one is "CD + Cholesky" which means the central difference method incorporating the Cholesky LDL^T decomposition method, another is "CD + MJGS" which means the central difference method incorporating the mixed Jacobi/Gauss-Seidel method. In Figure 5.11 through Figure 5.14 the relative speedup is defined as the execution time of CD + Cholesky on one processor divided by the execution time of CD + MJGS, and relative processor utilization is defined as the ratio of relative speedup to number of processors. The forward-backward substitutions in the experiments are performed sequentially since their parallelism is very poor.

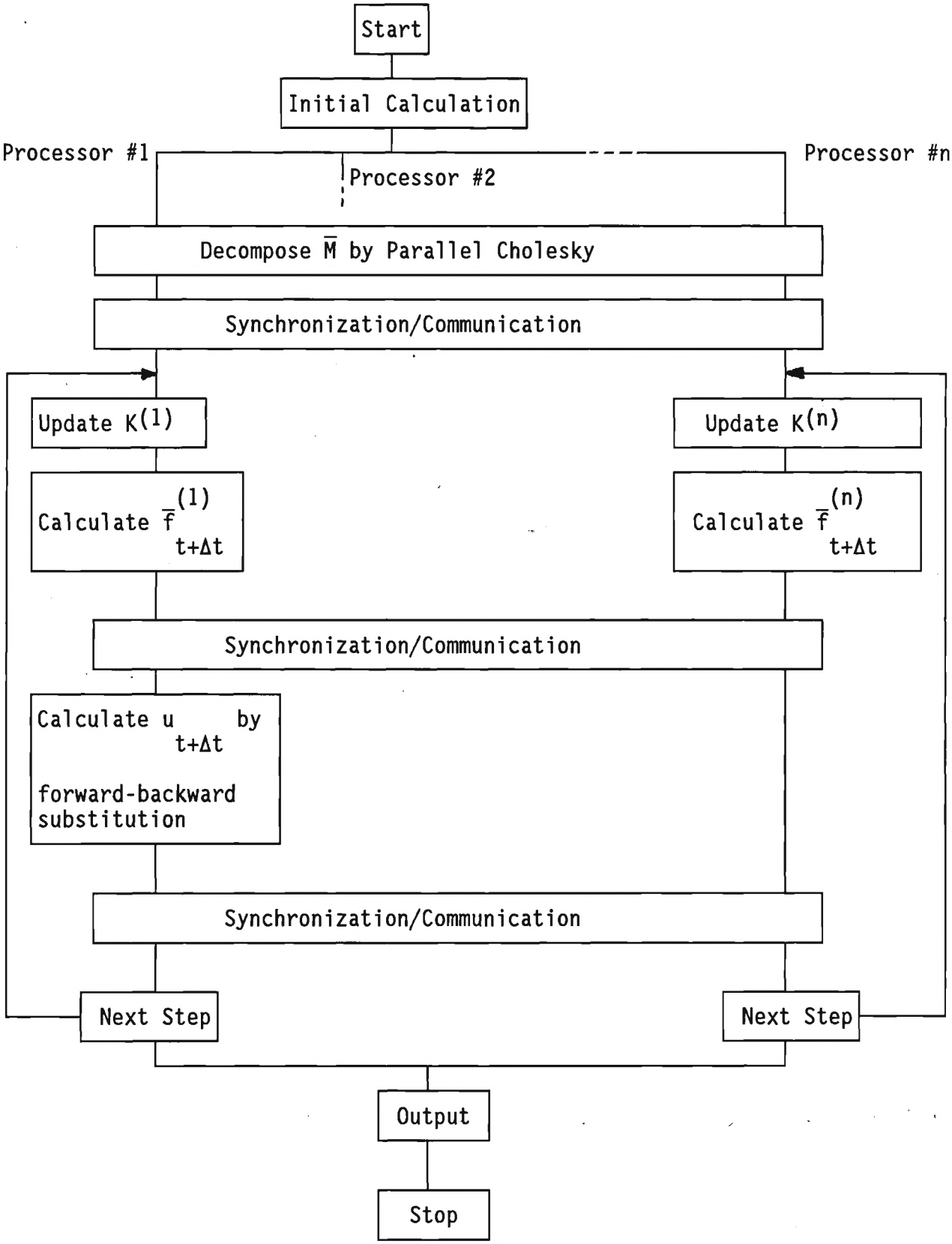


Figure 5.7. The Parallel Central Difference Method Incorporating Parallel Cholesky.

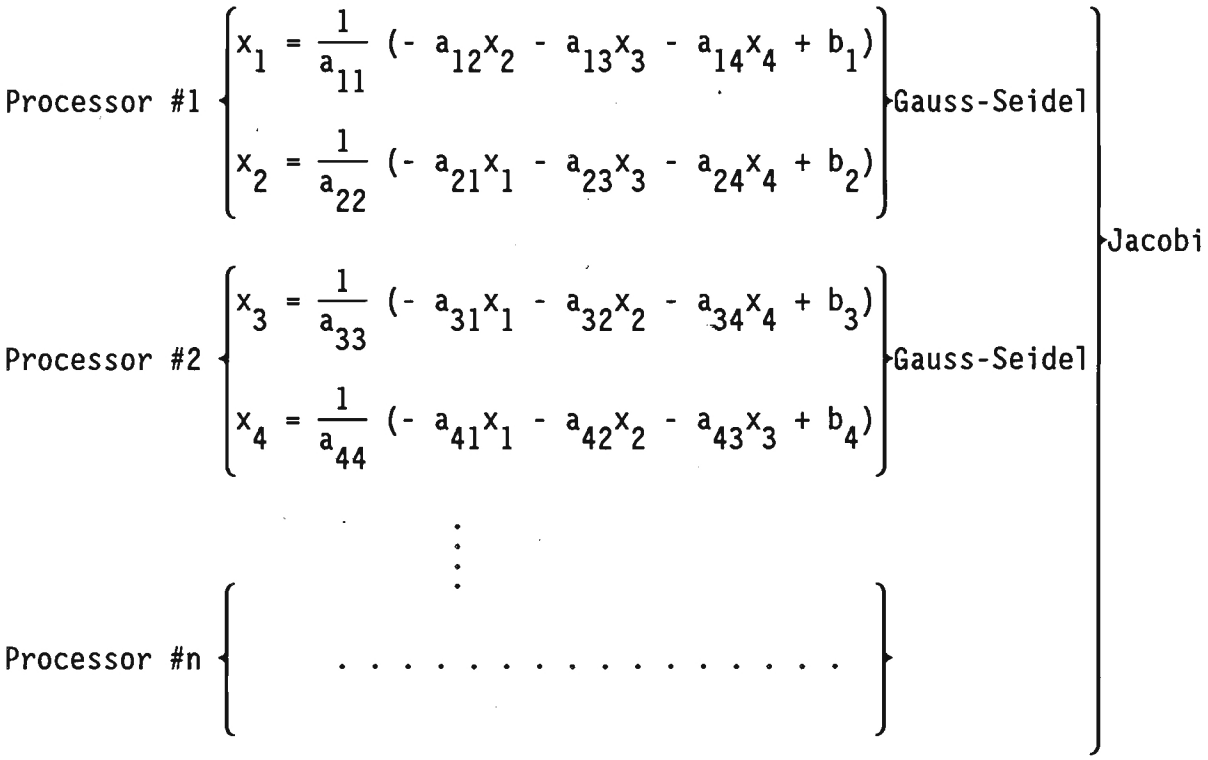


Figure 5.8. The Mixed Jacobi/Gauss-Seidel Method.

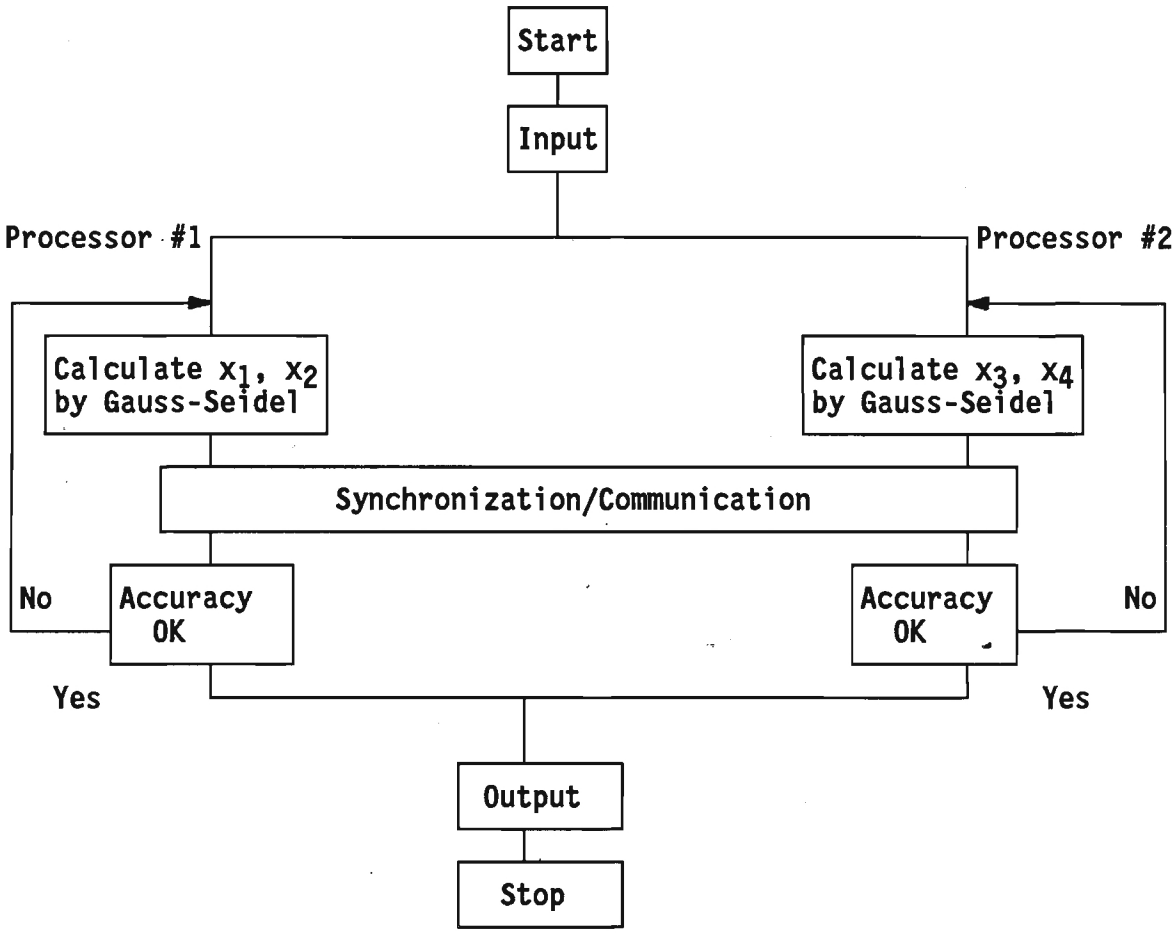


Figure 5.9 Parallel Implementation of the Mixed Jacobi/Gauss-Seidel Method.

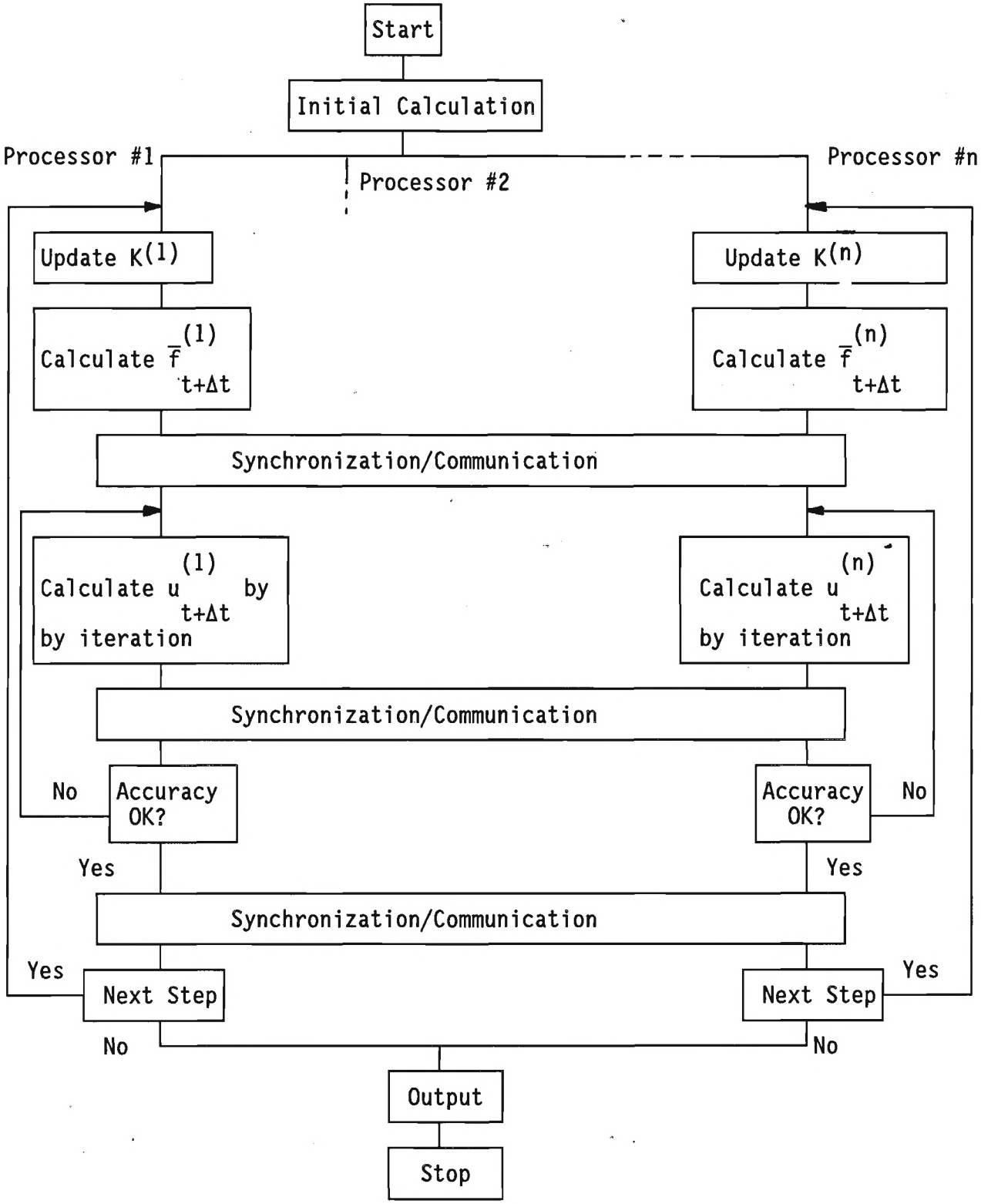


Figure 5.10. The Parallel Central Difference Method Incorporating the Parallel MJGS Method.

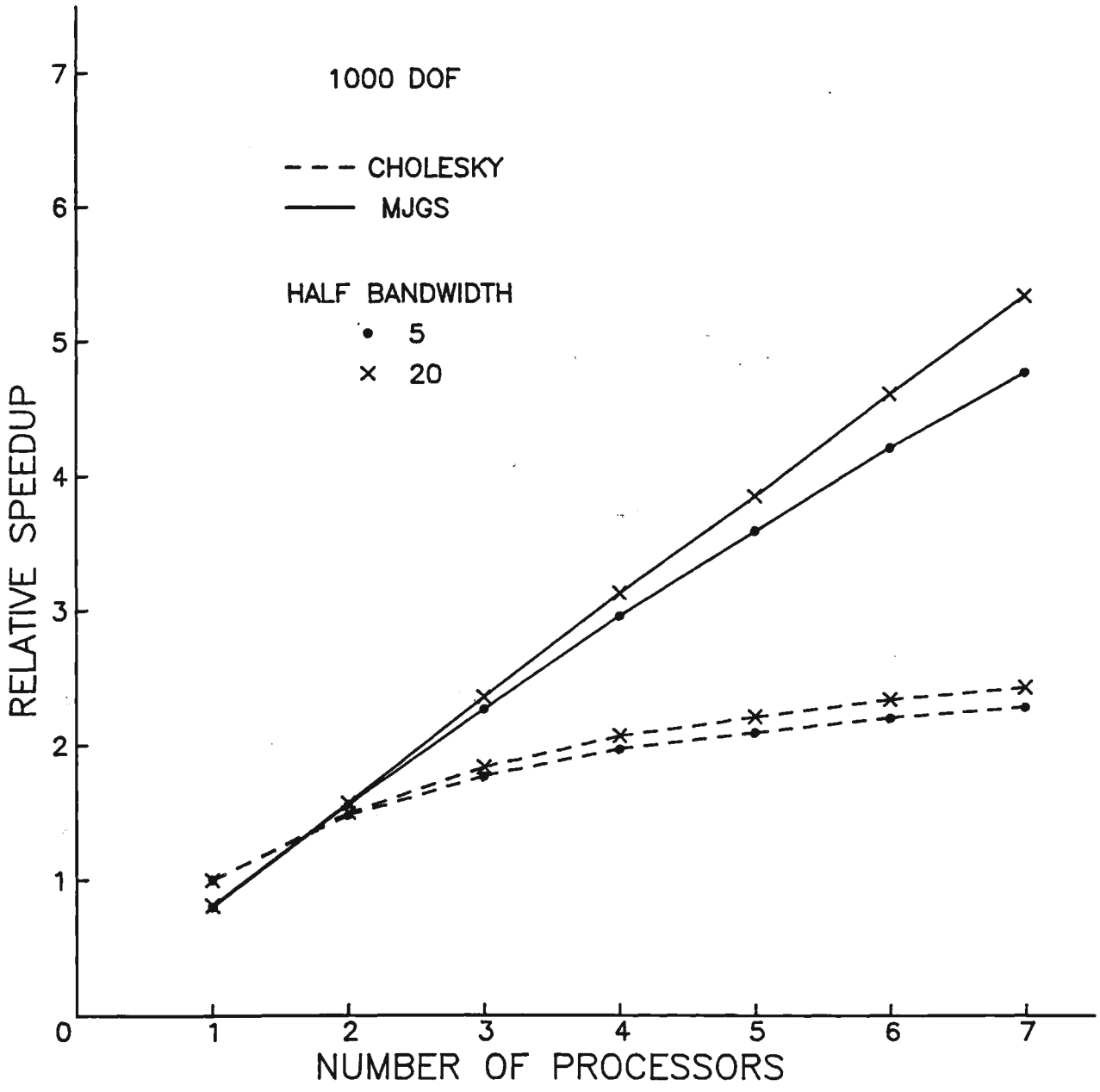


Figure 5.11. Cholesky Versus MJGS When Incorporated in the Central Difference Method. Nondiagonal Mass Mtrix. 100 Time Steps.

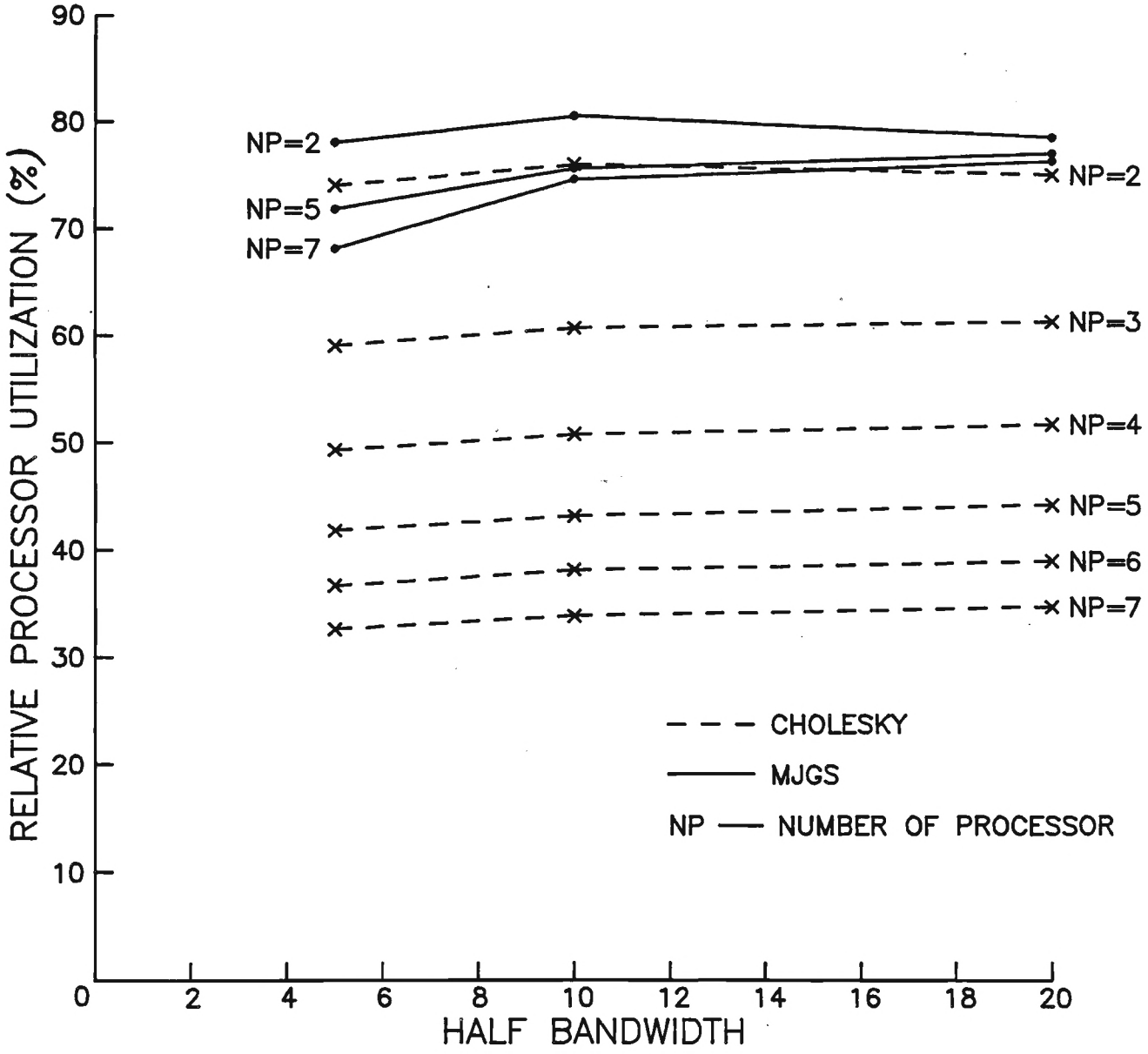


Figure 5.12. Cholesky Versus MJGS When Incorporated in the Central Difference Method. Nondiagonal Mass Matrix. 100 Time Steps.

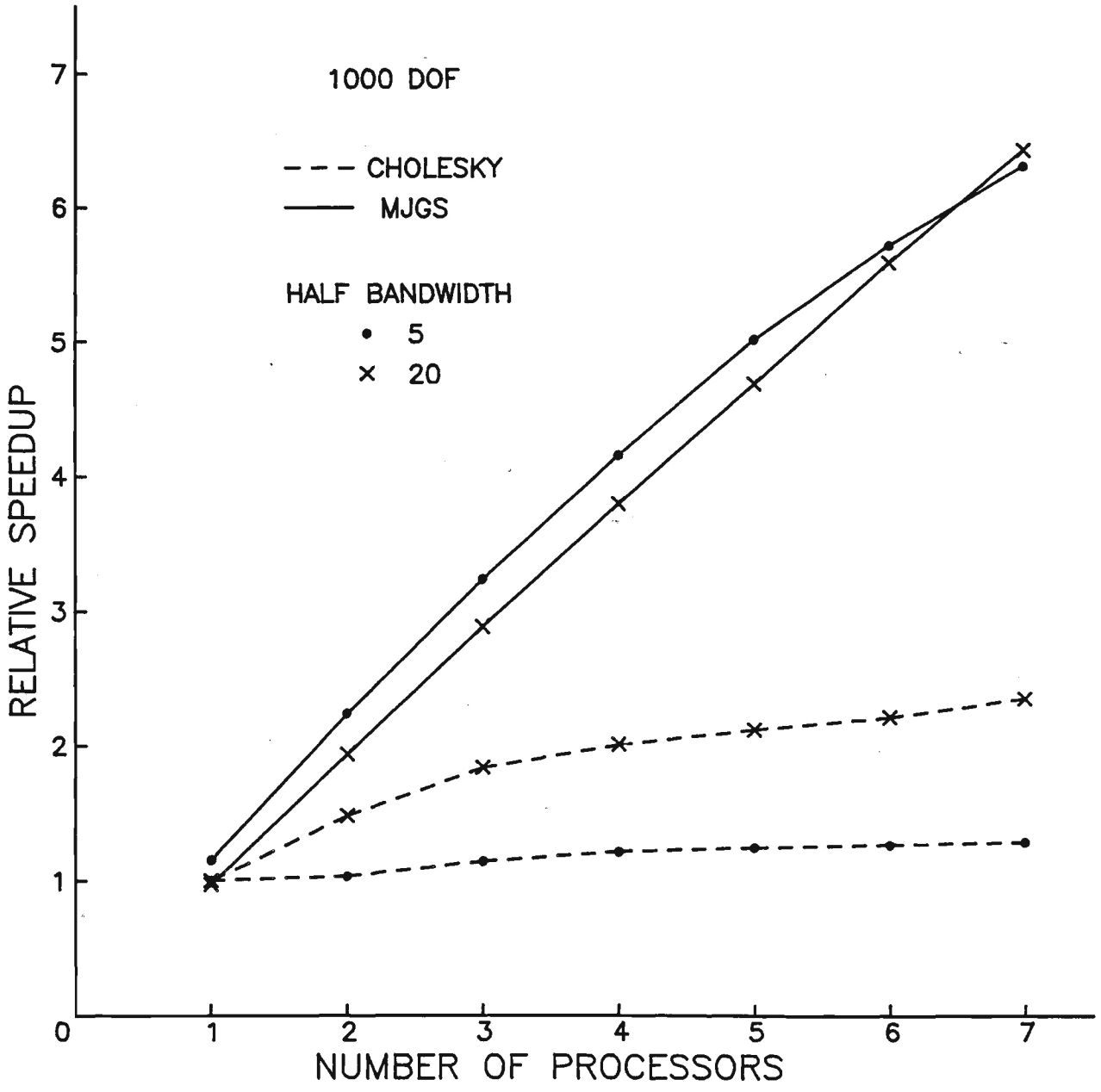


Figure 5.13. Cholesky Versus MJGS When Incorporated in the Central Difference Method. Nondiagonal Mass Matrix. 5 Time Steps.

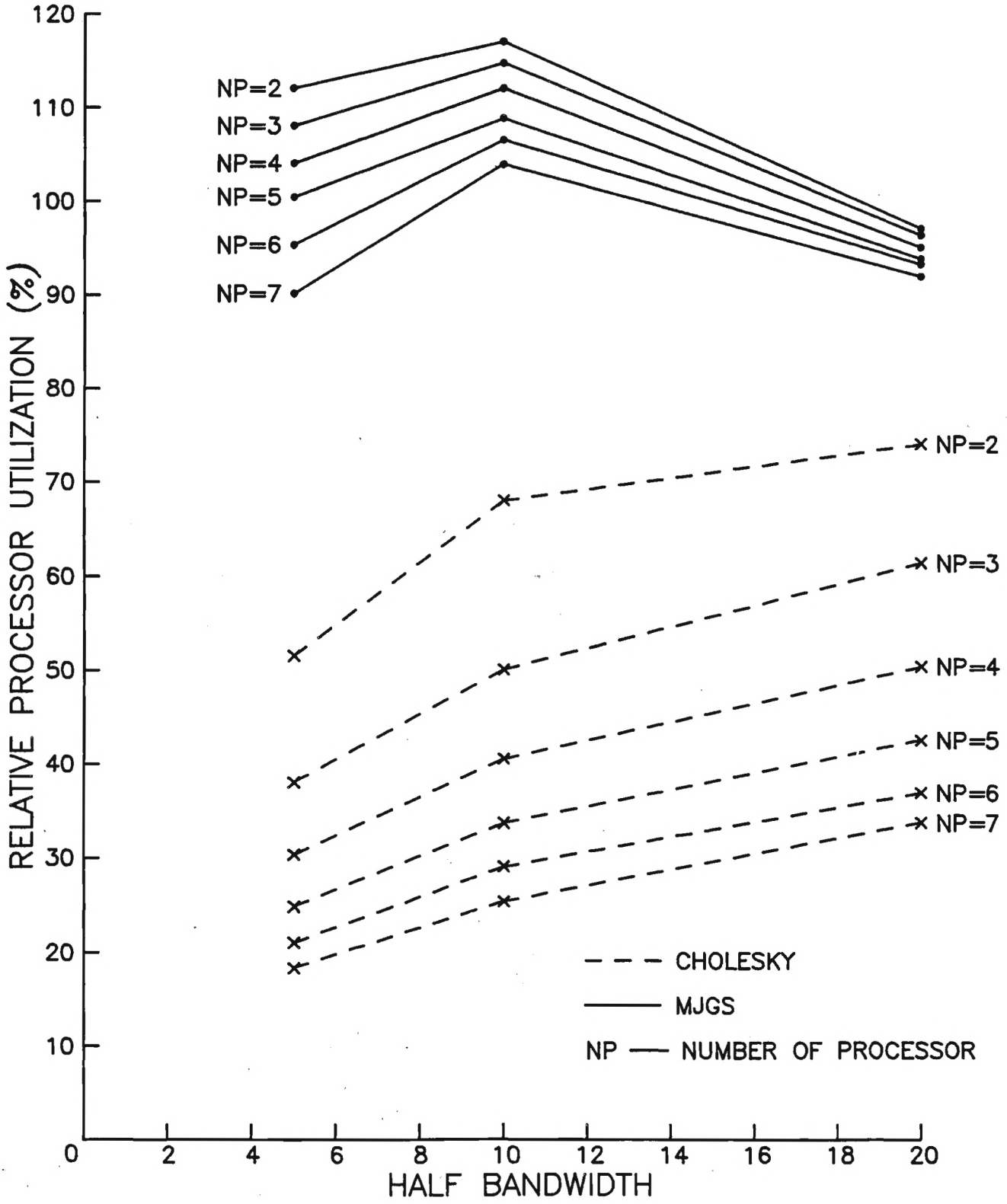


Figure 5.14. Cholesky Versus MJGS When Incorporated in the Central Difference Method. Nondiagonal Mass Matrix. 5 Time Steps.

The first point that can be seen from the figures is that the parallel effectiveness of CD + MJGS is much higher than CD+ Cholesky. Therefore, the overall efficiency of CD + MJGS is much higher than CD + Cholesky when the number of processors is large (say, 7) although the sequential computational efficiencies of the two methods are similar.

Another point is that when the number of integration step is greater, the parallel effectiveness of CD + Cholesky for small bandwidth is higher and the effect of bandwidth on the overall parallel effectiveness is less evident. The reason is that only one decomposition of the effective mass matrix is needed for the whole integration procedure in the case where the mass and damping matrices are constant. Therefore, the larger the number of integration step taken, the smaller the influence of the ineffectiveness of parallel Cholesky decomposition for small bandwidth on the overall parallel effectiveness of integration.

An important factor which make the parallel effectiveness of CD + Cholesky relatively poor is that the parallel effectiveness of the forward - backward substitution is low and yet it has to be involved at each time step. To understand this effect, suppose that the forward - backward substitution is performed sequentially in the integration procedures and let r be the ratio of the time taken to perform this sequential forward - backward substitution to the time taken to perform the rest of computation of integration. Also suppose that the speedup in performing the rest of computation is "ideal" and 7 processors are used. Then the overall speedup is

$$\text{speedup} = (1 + r)/(1/7 + r).$$

This relationship is shown in Figure 5.15. From the figure it can be seen that the negative effect of sequential execution of forward - backward substitution on the overall speedup is significant if r is large.

Another noticeable issue is the diagonal dominance of the coefficient matrix. It is well known that it effects the rate of convergence and the efficiency. In the worst case the computation does not converge. Therefore, attention should be paid to this issued when the iterative method is used in practice.

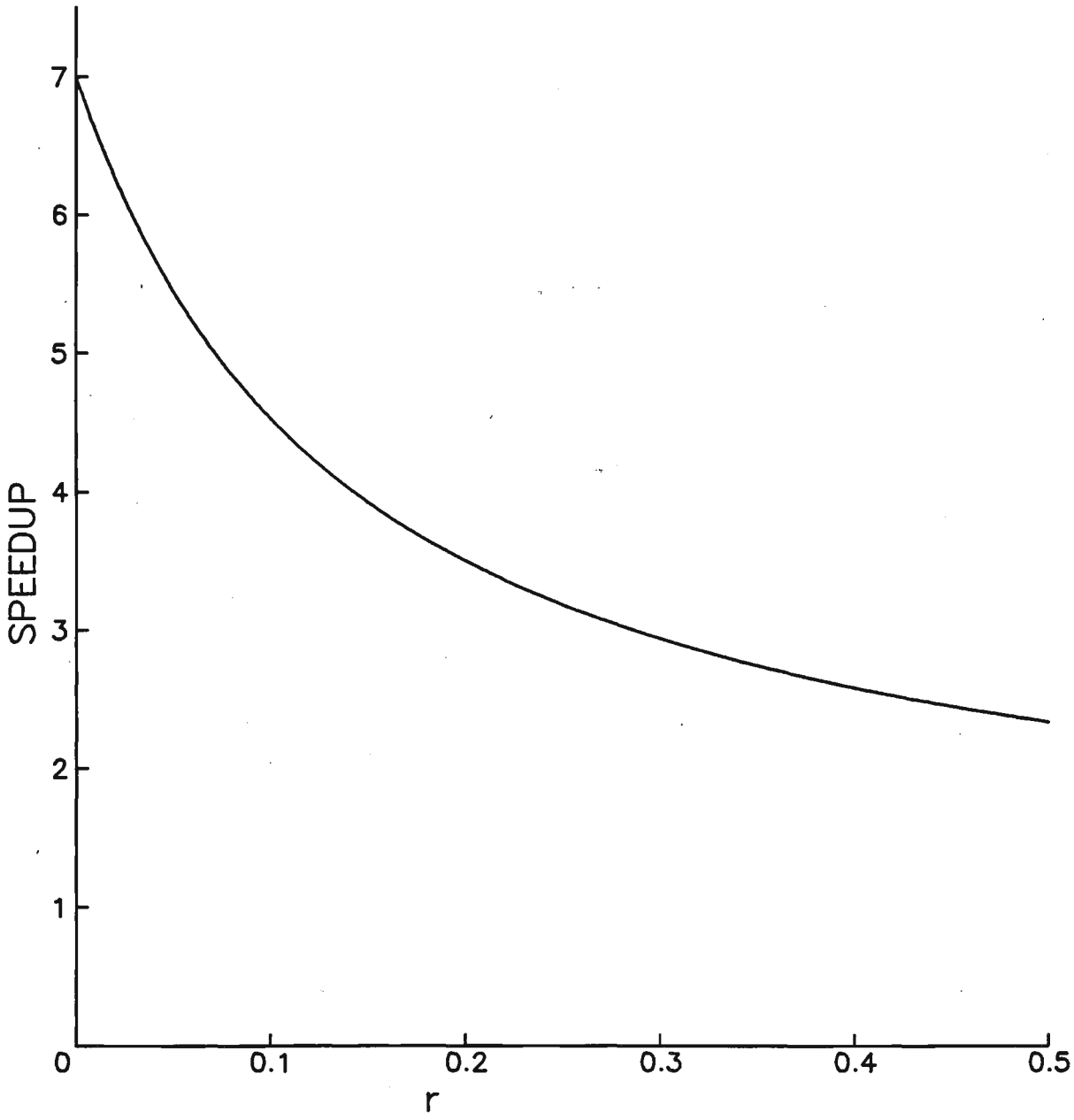


Figure 5.15. The Effects of r on Speedup.

5.6 PARALLEL IMPLEMENTATION OF PREDICTOR-CORRECTOR METHODS

The predictor-corrector methods are less popular than the Newmark and central difference methods in finite element codes. But the initial studies show that the methods have good parallel effectiveness in the case of diagonal mass matrix. Therefore they are attractive and worth investigating. As the first stage of the research in this area, the Adams-Moulton method and the Kunz method are chosen to be studied because they have good stability and efficiency.

5.6.1. The Adams-Moulton Method

The Adams-Moulton method [5.9] is a predictor-corrector method which uses the following predictor:

$$u_{t+\Delta t} = u_t + \frac{\Delta t}{24}\left(55\dot{u}_t - 59\dot{u}_{t-\Delta t} + 37\dot{u}_{t-2\Delta t} - 9\dot{u}_{t-3\Delta t}\right) \tag{5.17}$$

$$\dot{u}_{t+\Delta t} = \dot{u}_t + \frac{\Delta t}{24}\left(55\ddot{u}_t - 59\ddot{u}_{t-\Delta t} + 37\ddot{u}_{t-2\Delta t} - 9\ddot{u}_{t-3\Delta t}\right) \tag{5.18}$$

and the following corrector:

$$u_{t+\Delta t} = u_t + \frac{\Delta t}{24}\left(9\dot{u}_{t+\Delta t} + 19\dot{u}_t - 5\dot{u}_{t-\Delta t} + \dot{u}_{t-2\Delta t}\right) \tag{5.19}$$

$$\dot{u}_{t+\Delta t} = \dot{u}_t + \frac{\Delta t}{24}\left(9\ddot{u}_{t+\Delta t} + 19\ddot{u}_t - 5\ddot{u}_{t-\Delta t} + \ddot{u}_{t-2\Delta t}\right) \tag{5.20}$$

The values of $\ddot{u}_{t+\Delta t}$ used in Equation (5.20) are calculated from Equation (5.2) by use of $\dot{u}_{t+\Delta t}$ and $u_{t+\Delta t}$ obtained from the predictor. Therefore, equation solving is needed when the mass matrix is nondiagonal. The method assumes a set of starting values already calculated by some other technique. It has local errors of $O(\Delta t^5)$ and global errors of $O(\Delta t^4)$. Here local error means the error for one step only and global error means the accumulated error over many steps. Reference [5.9] shows that the method is a stable method and this is an important advantage of the method over some other predictor-corrector methods such as the Milne method.

The method is implemented in parallel to solve the test problem shown in Figure 5.1. The flowchart of implementation and the computation results are shown in Figure 5.16 and Figure 5.17. From the flowchart and Equation 5.17 through 5.20 it can be seen that the parallelism of the algorithm is good when the mass matrix is diagonal. In the case of nondiagonal mass matrix the parallelism is related to the equation solving method used. The situation is somewhat similar to that of the central difference method.

5.6.2. The Kunz Method

In the case where the damping coefficients are zero and it is not necessary to find the velocities, the Kunz method [5.9] can be used. The method uses the following predictor:

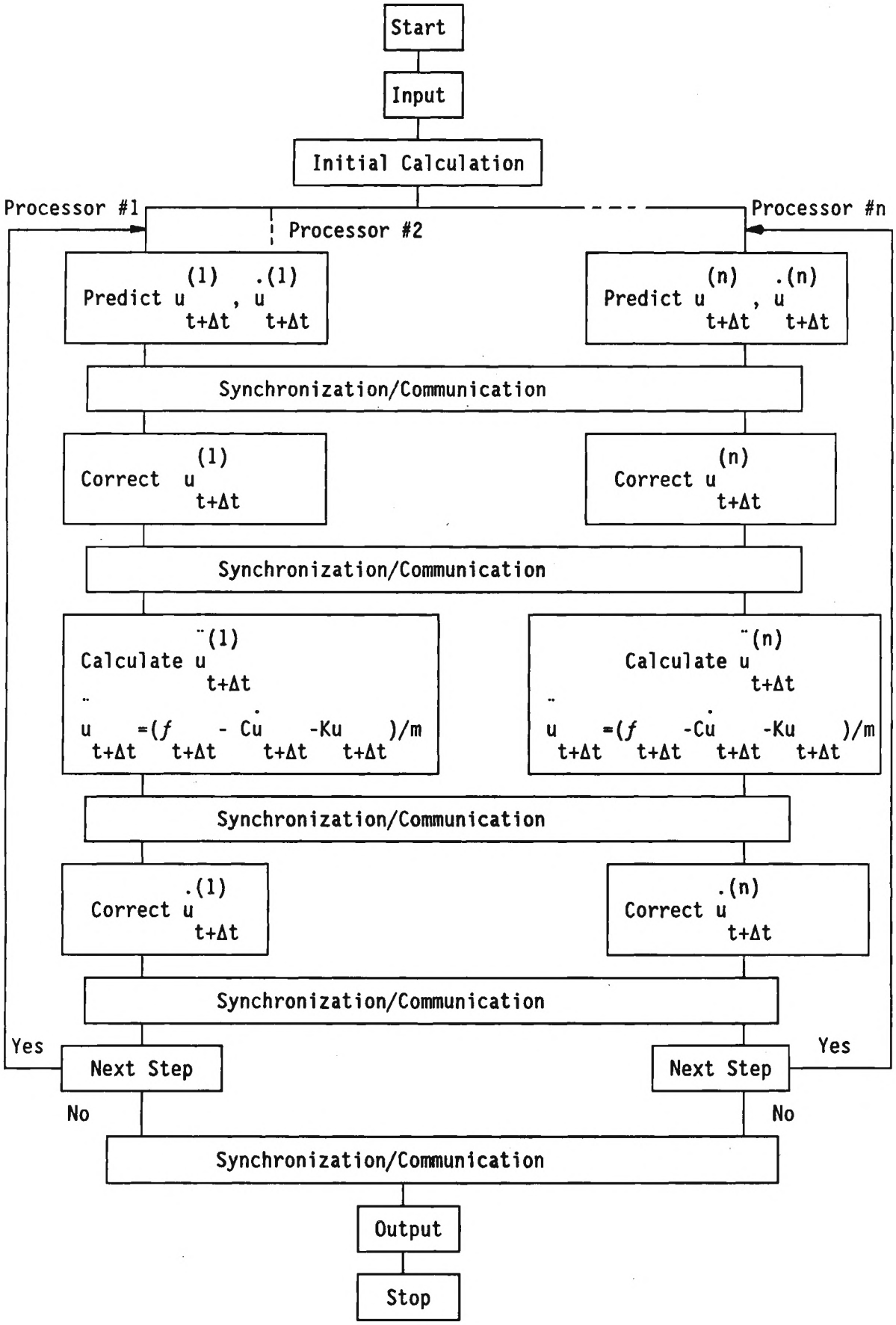
$$u_{t+\Delta t} = u_t + u_{t-2\Delta t} - u_{t-3\Delta t} + \frac{\Delta t^2}{4}\left(5\ddot{u}_t + 2\ddot{u}_{t-\Delta t} + 5\ddot{u}_{t-2\Delta t}\right)$$

and the following corrector:

$$u_{t+\Delta t} = 2u_t - u_{t-\Delta t} + \frac{\Delta t^2}{12}\left(\ddot{u}_{t+\Delta t} + 10\ddot{u}_t + \ddot{u}_{t-\Delta t}\right) \qquad (5.21)$$

Similarly, the values of $\ddot{u}_{t+\Delta t}$ used in (5.21) are calculated from Equation (5.2) by use of $u_{t+\Delta t}$ obtained from the predictor, and equation solving is required in the case of nondiagonal mass matrix. The method has also local errors of $O(\Delta t^5)$ and global errors of $O(\Delta t^4)$.

The Kunz method is also implemented in parallel to solve the test problem in Figure 5.1. Figure 5.18 is the flowchart and Figure 5.19 shows the computation results.


Start
Input
Initial Calculation
Processor #1
Processor #2
Processor #n
Predict u(1)t+Δt , u.(1)t+Δt
Predict u(n)t+Δt , u.(n)t+Δt
Synchronization/Communication
Correct u(1)t+Δt
Correct u(n)t+Δt
Synchronization/Communication
Calculate ü(1)t+Δt
ü t+Δt =(f t+Δt - Cu̇ t+Δt -Ku t+Δt)/m
Calculate ü(n)t+Δt
ü t+Δt =(f t+Δt -Cu̇ t+Δt -Ku t+Δt)/m
Synchronization/Communication
Correct .(1)u t+Δt
Correct .(n)u t+Δt
Synchronization/Communication
Yes
Next Step
No
Yes
Next Step
No
Synchronization/Communication
Output
Stop

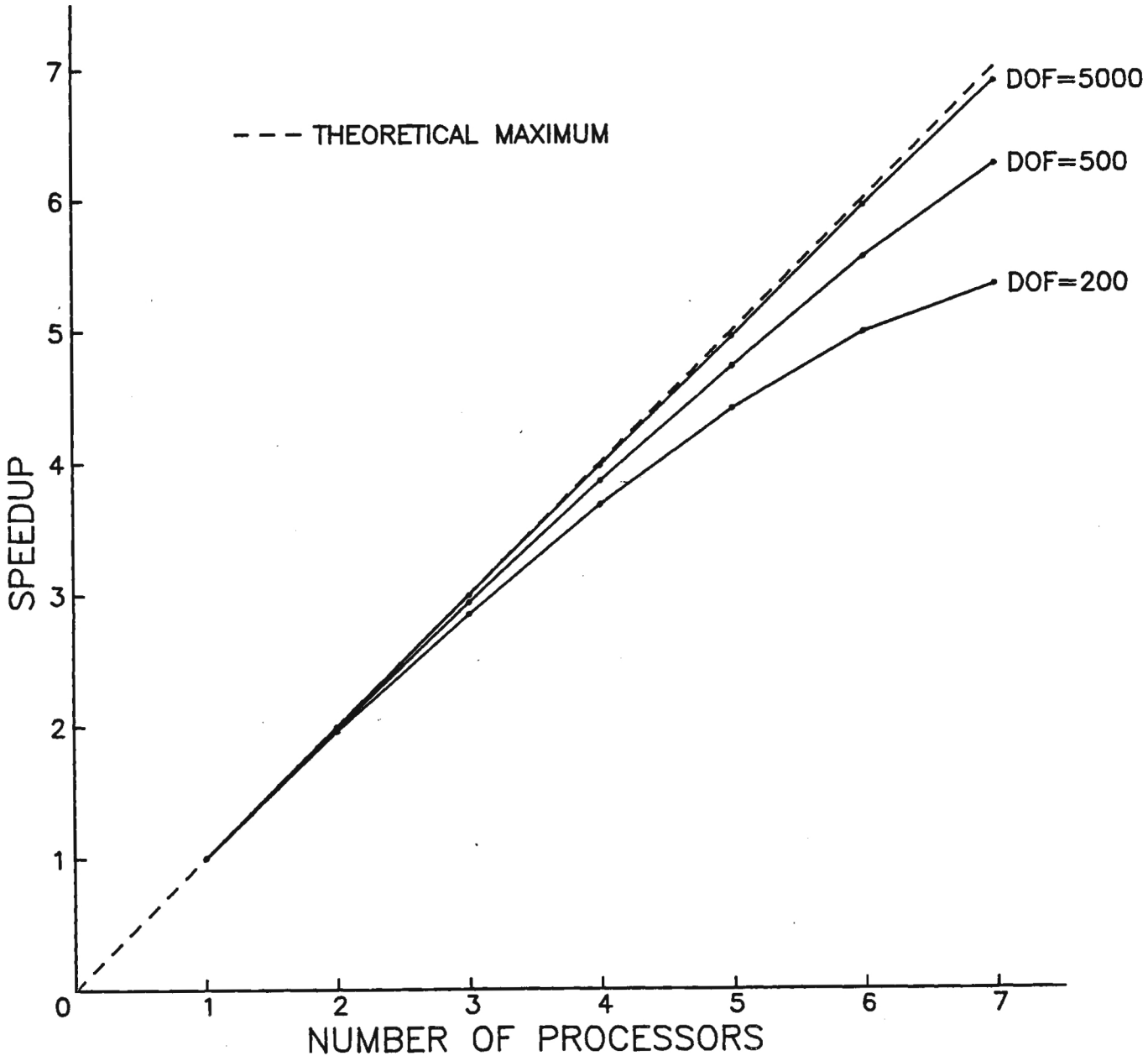


Figure 5.17. The Adams-Moulton Method.

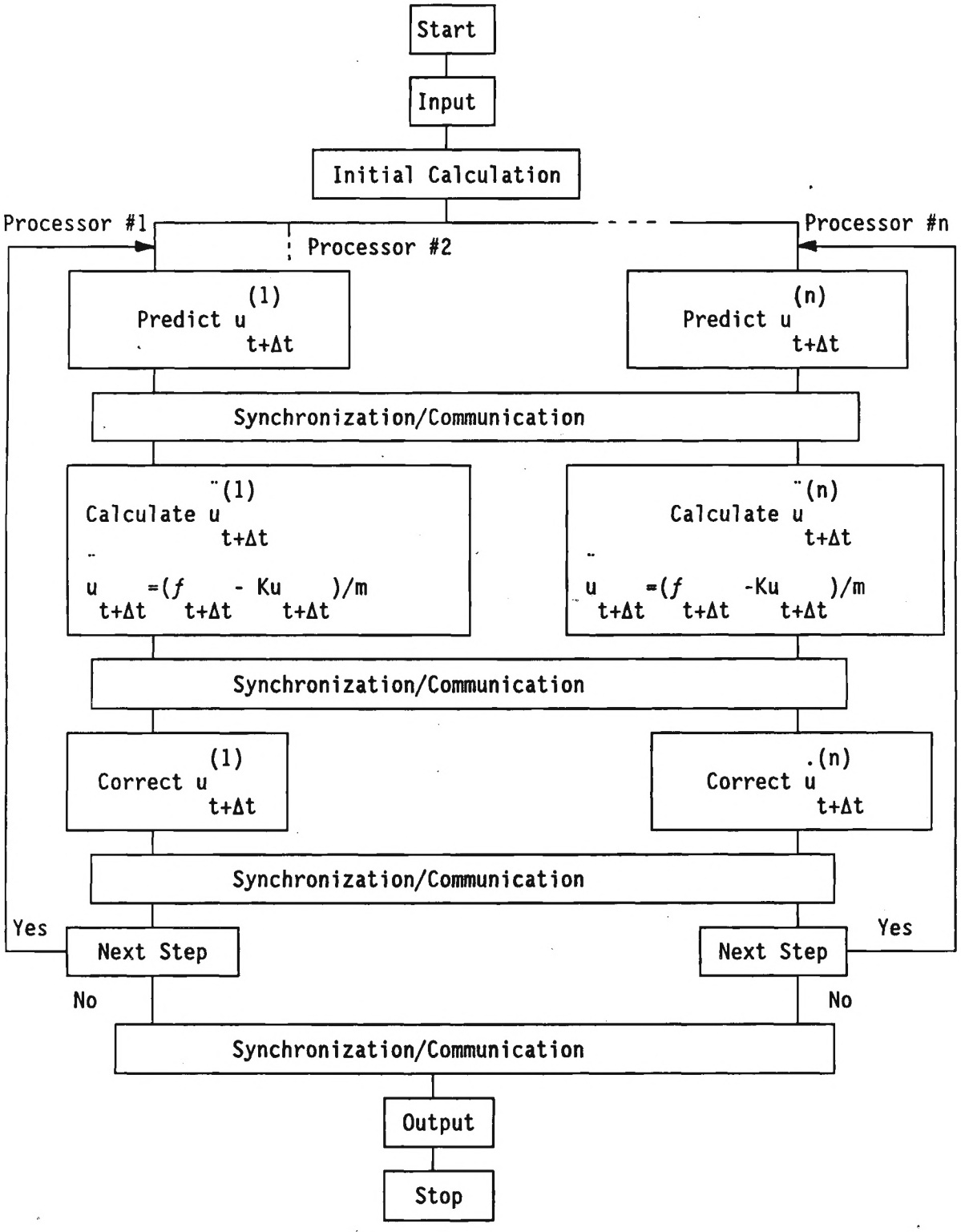


Figure 5.18. The Parallel Kunz Method.

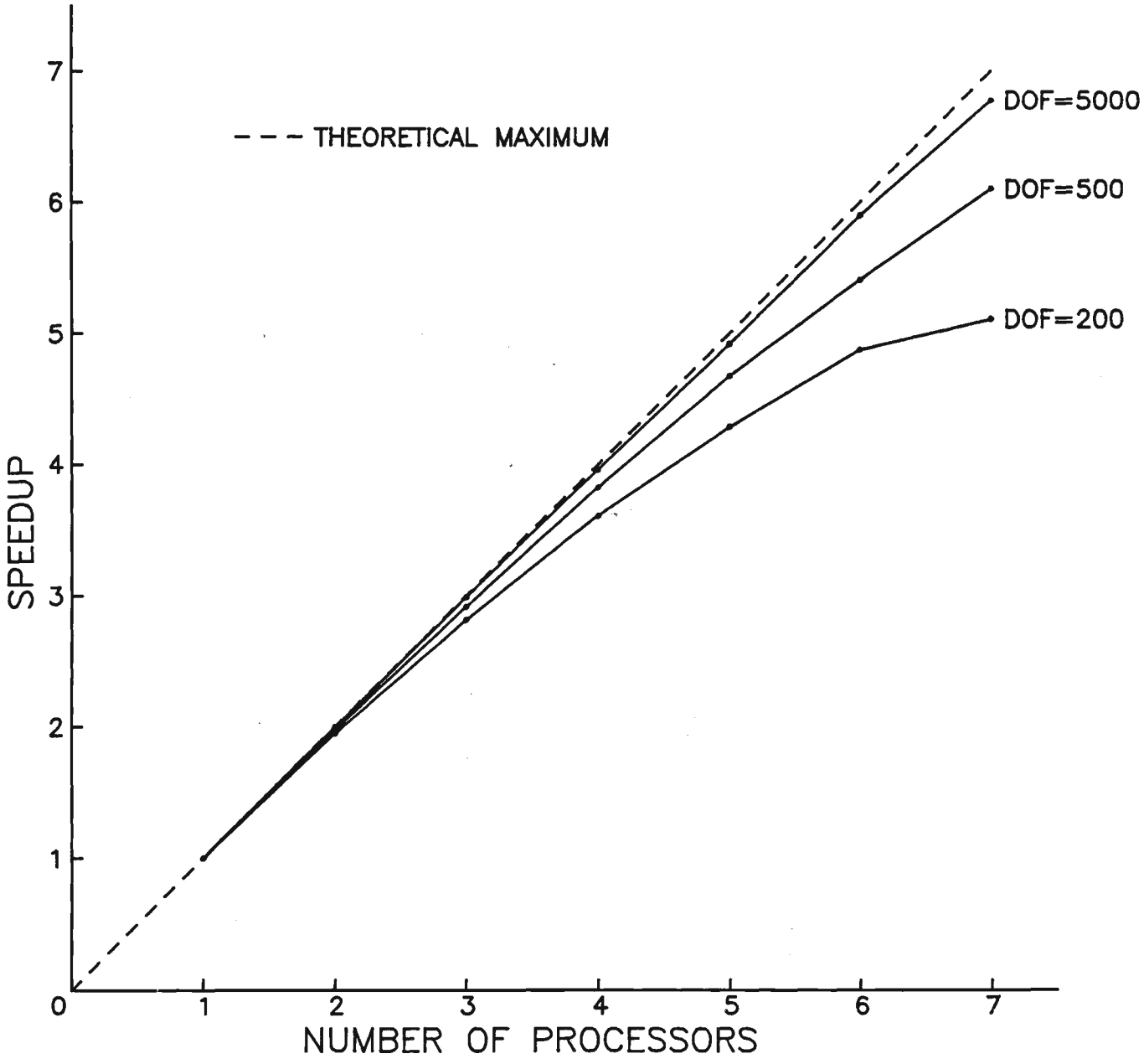


Figure 5.19. The Kunz Method.

The consideration of step size control of the two predictor-corrector methods and the comparison with other integration techniques remain as future work.

5.7 SOLVING A PLAIN STRESS TEST PROBLEM IN PARALLEL

Figure 5.20 shows another test problem used in the study. It is a plain stress problem corresponding to a cantilever beam subjected to in-plane loading. In Figure 5.20, S_1, S_2, ... S_9 are also nonlinear springs where the spring force is proportional to the cubic of the displacements.

The central difference method and the Newmark method are first used to solve the plain stress test problem in parallel and comparison between the two methods is made. Since a consistent mass model is used the mass matrix is nondiagonal and an equation solving strategy must be involved. Here the mixed Jacobi/Gauss-Seidel method is used to solve the sets of equations. Figure 5.21 and Figure 5.22 are the flowcharts of the implementation for the two methods respectively. Figure 5.23 and Figure 5.24 show the computation results for the case where 121 four-node axisymmetric elements are used. The speedups are better than those of the simple one-dimensional test problem of similar size because more computations are involved. Figure 5.25 is the comparison between the central difference and the Newmark methods. From the figure it is seen that the two methods are similar in both computational efficiency and parallel effectiveness since the central difference method also requires solution to simultaneous equations in the case of nondiagonal mass matrix. Therefore, as mentioned in Section 5.4, the central difference method is not recommended in such a case because of its conditional stability.

The studies are continuing and the implementation of other parallel integration methods and related evaluation are to be performed.

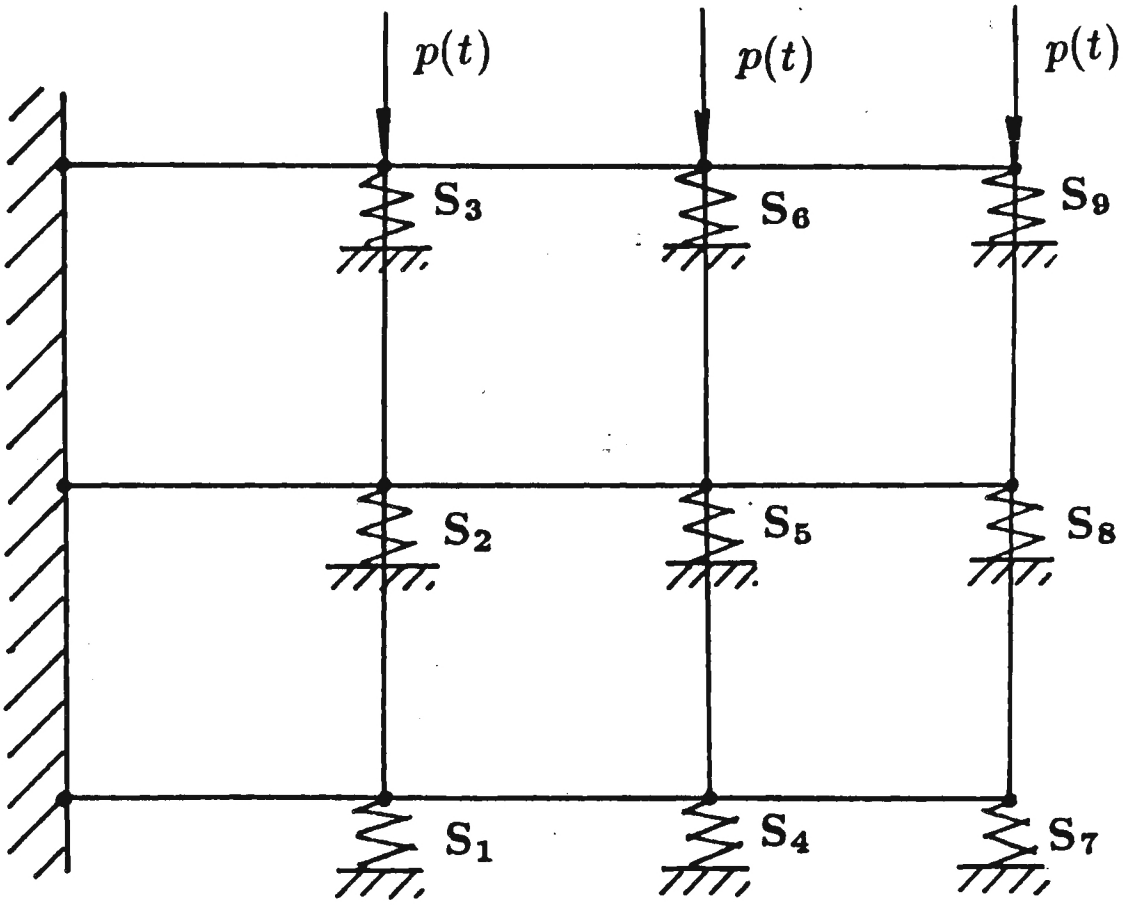


Figure 5.20. A Plane Stress Problem.

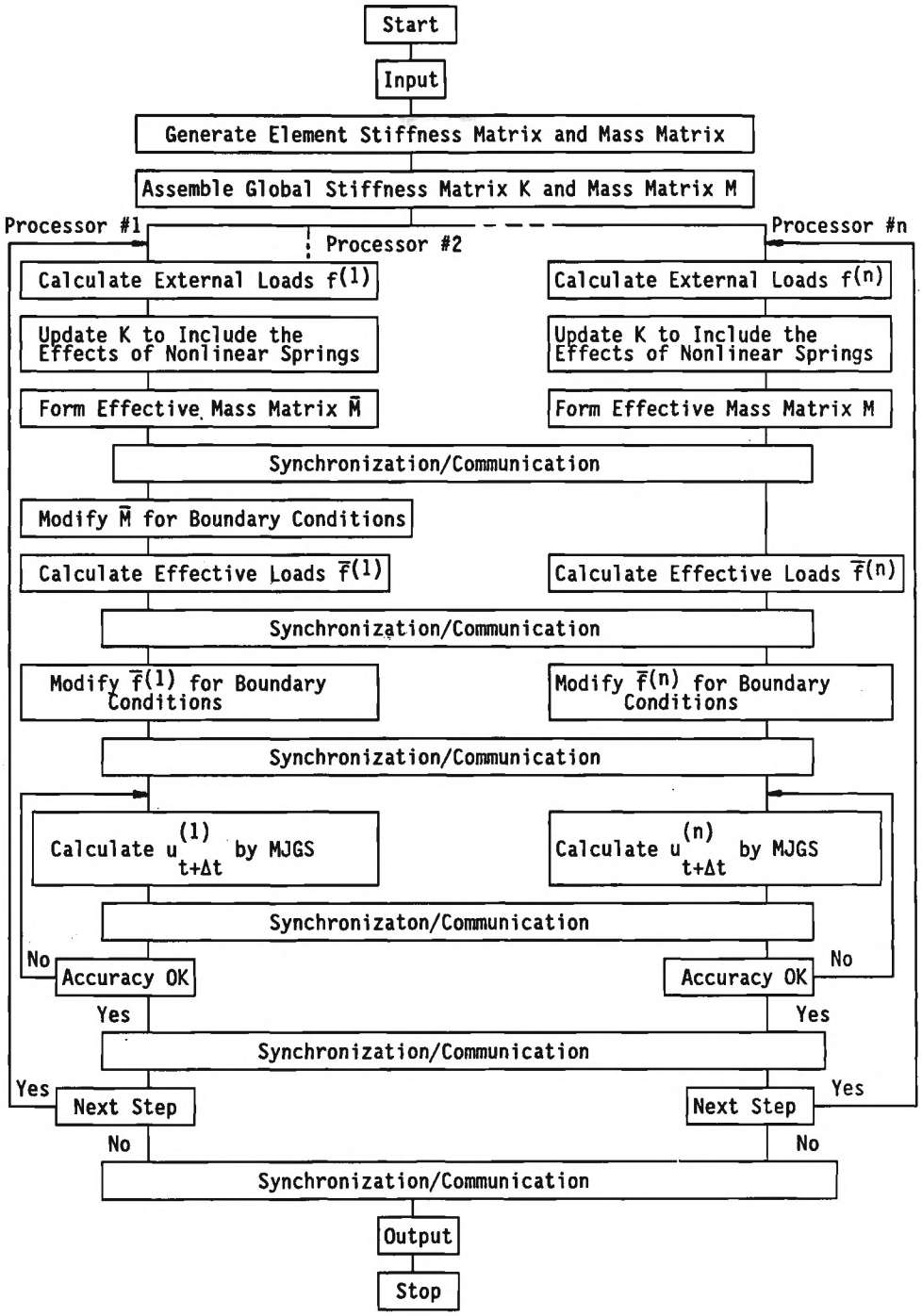
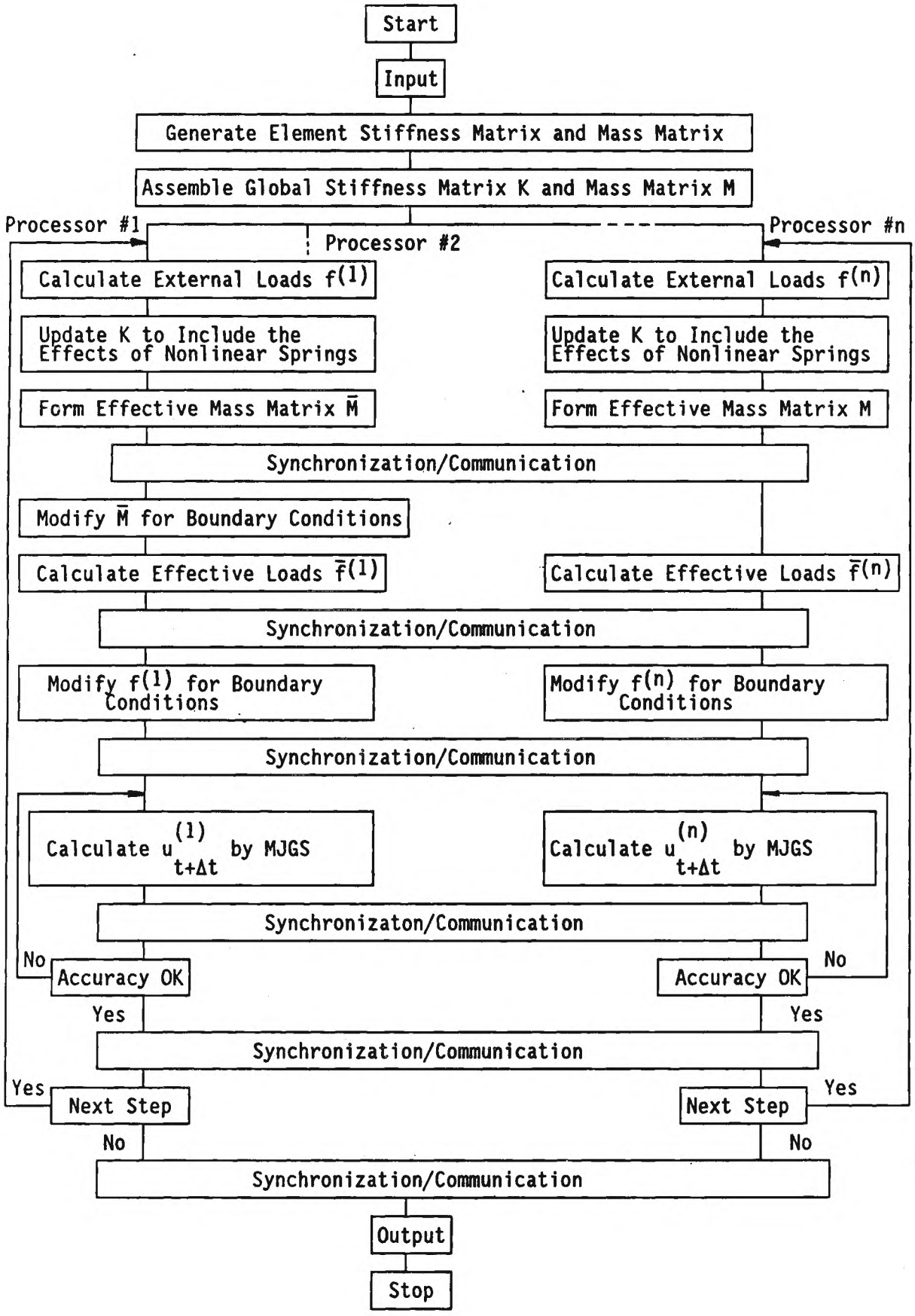


Figure 5.21. Parallel Central Difference + MJGS for the Plane Stress Problem.


Start
Input
Generate Element Stiffness Matrix and Mass Matrix
Assemble Global Stiffness Matrix K and Mass Matrix M
Processor #1
Processor #2
Processor #n
Calculate External Loads f(1)
Calculate External Loads f(n)
Update K to Include the Effects of Nonlinear Springs
Update K to Include the Effects of Nonlinear Springs
Form Effective Mass Matrix M̄
Form Effective Mass Matrix M
Synchronization/Communication
Modify M̄ for Boundary Conditions
Calculate Effective Loads f̄(1)
Calculate Effective Loads f̄(n)
Synchronization/Communication
Modify f(1) for Boundary Conditions
Modify f(n) for Boundary Conditions
Synchronization/Communication
Calculate u(1) t+Δt by MJGS
Calculate u(n) t+Δt by MJGS
Synchronizaton/Communication
No
Accuracy OK
Accuracy OK
No
Yes
Yes
Synchronization/Communication
Yes
Next Step
Next Step
Yes
No
No
Synchronization/Communication
Output
Stop

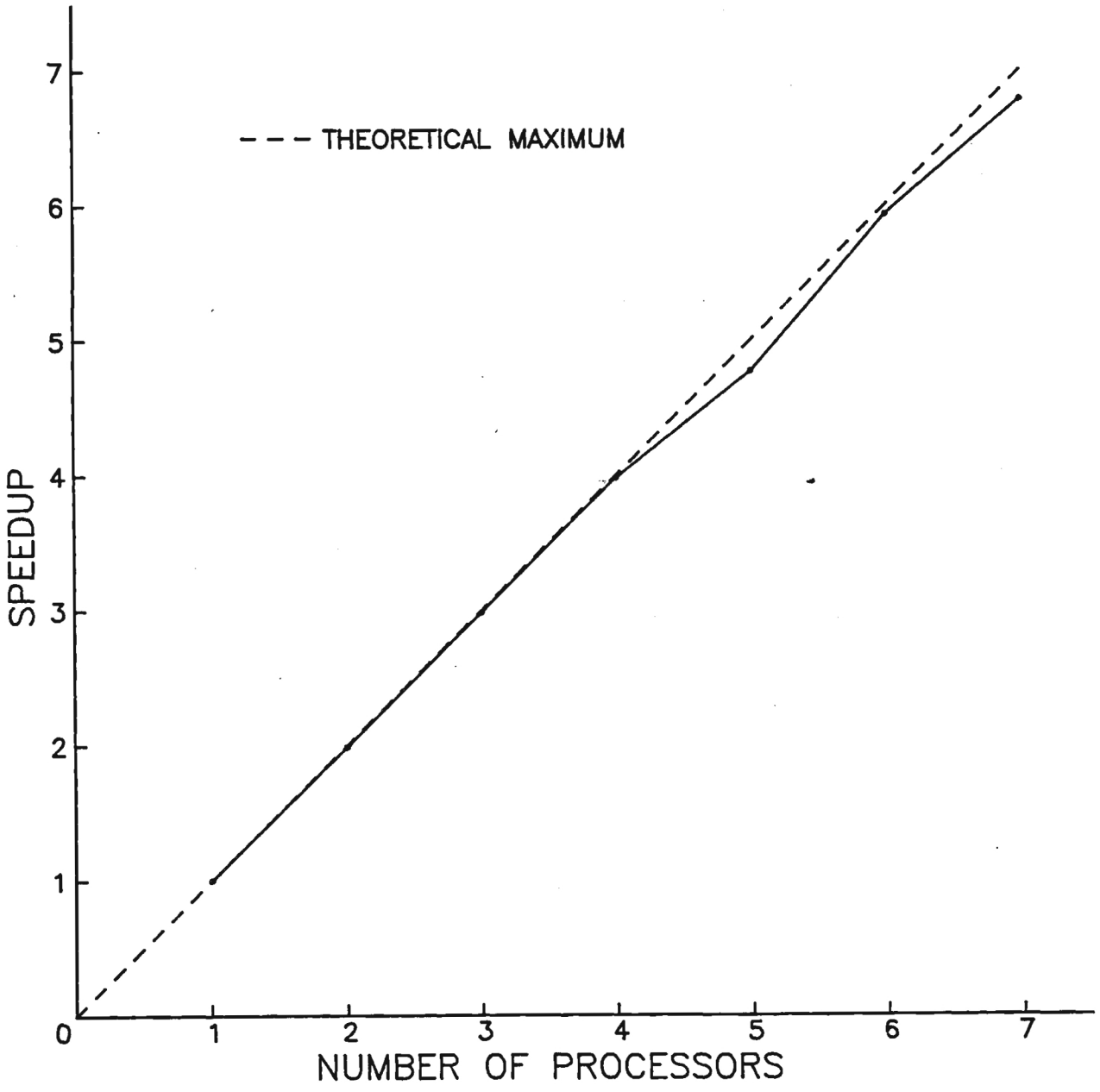


Figure 5.23. Parallel Central Difference + MJGS for the Plane Stress Problem.

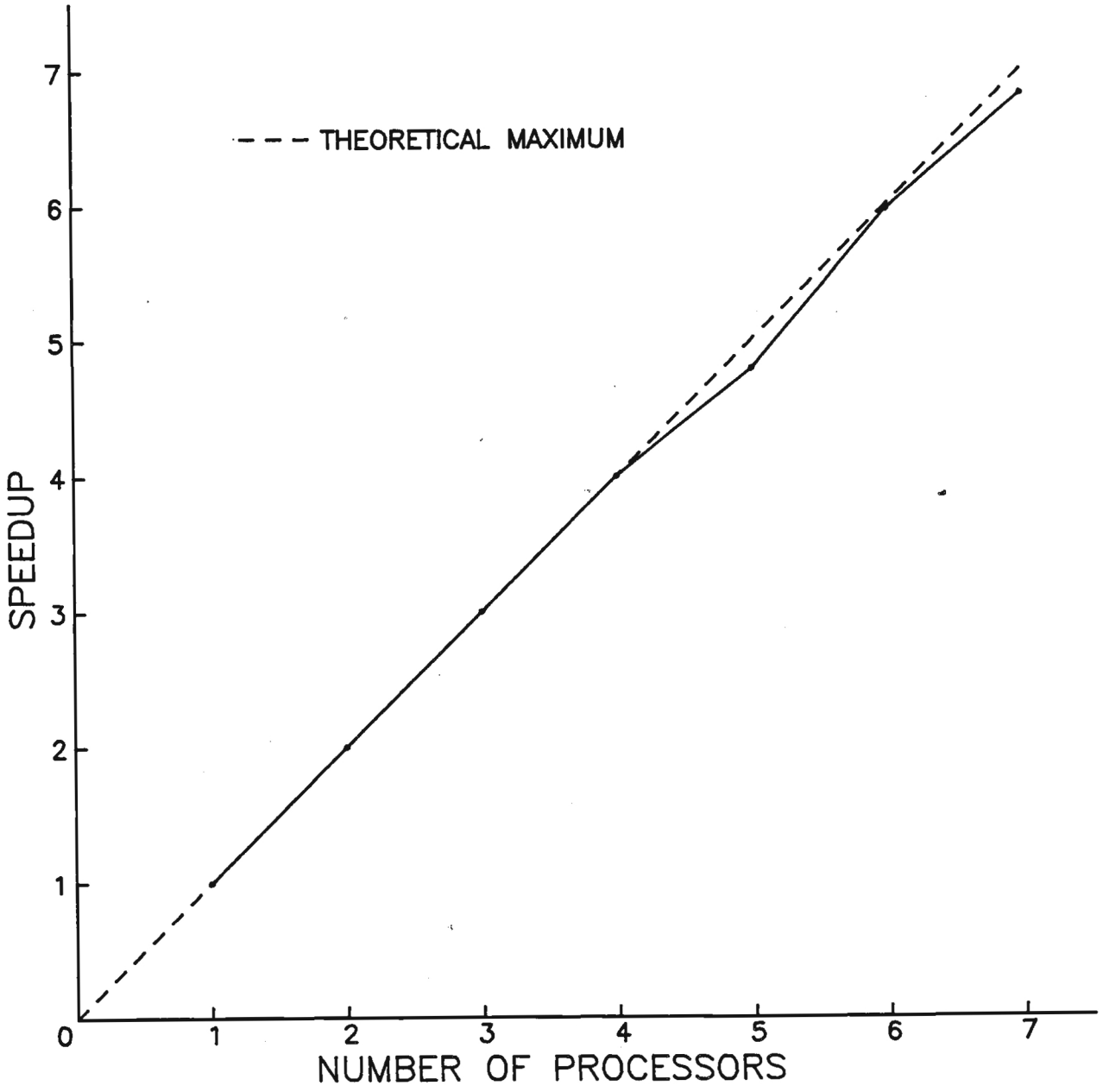


Figure 5.24. Parallel Newmark + MJGS for the Plane Stress Problem.

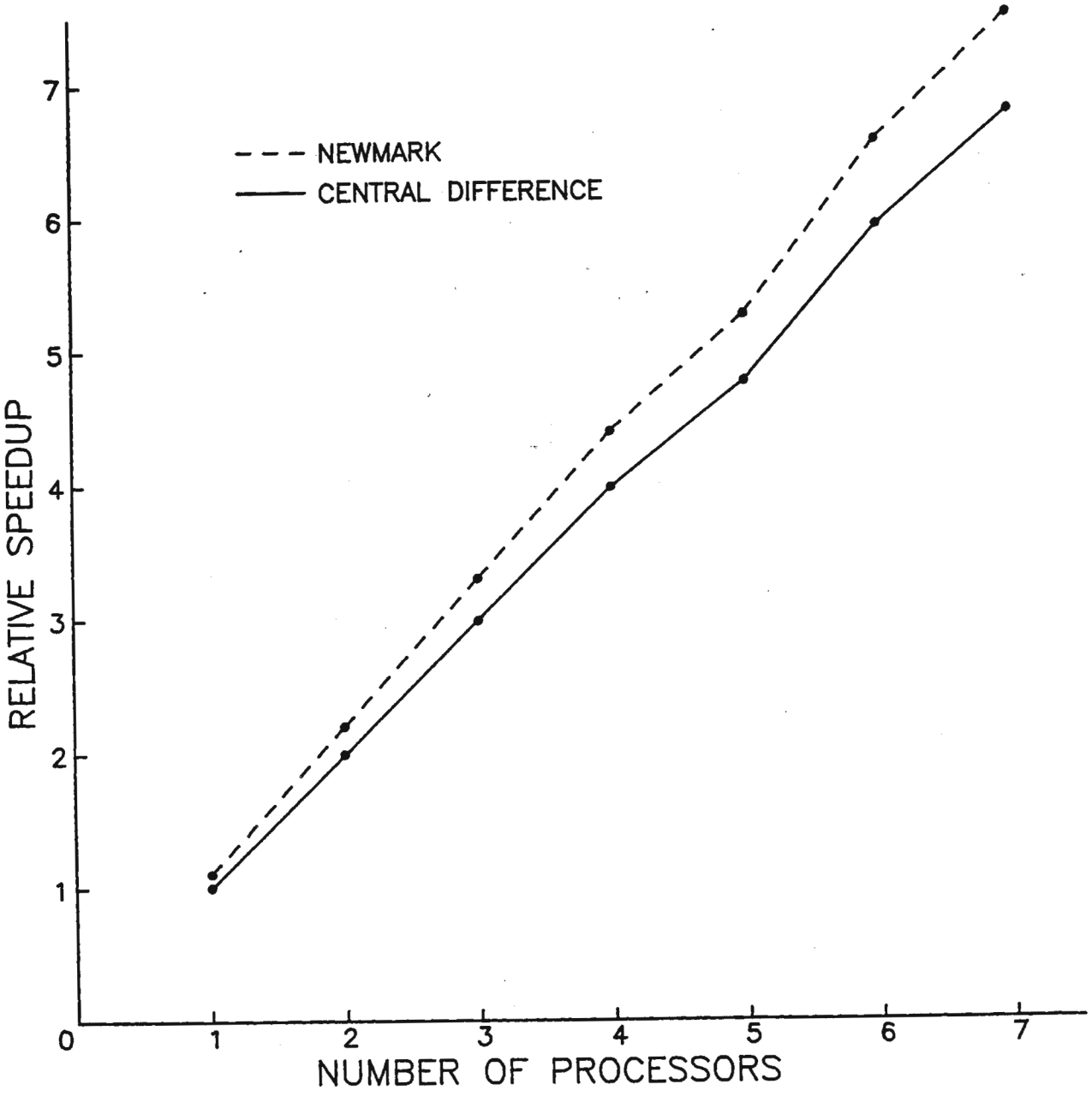


Figure 5.25. The Central Difference Method vs. the Newmark Method.

5.8 CONCLUDING REMARKS

The criteria for evaluating the robustness of parallel numerical integration algorithms are discussed and several commonly used methods - the central difference, Newmark, Houbolt and Wilson theta methods are examined. Some numerical experiments are performed to compare the explicit central difference method with the implicit methods incorporating cyclic reduction for the case of diagonal mass matrix. The results show that the central difference method has higher computational efficiency and parallel effectiveness in such a case and is recommended. In the case of nondiagonal mass matrix, the parallel effectiveness of the central difference method is similar to that of the implicit methods and is not recommended in general because of its conditional stability.

A comparison between the parallel Cholesky LDL^T decomposition and the mixed Jacobi/Gauss-Seidel (MJGS) method is also made when they are incorporated in the parallel central difference method for the case of nondiagonal mass matrix. The results show that the MJGS method has much higher parallel effectiveness than the Cholesky decomposition method when incorporated in the integration procedure. The results which are based on a limited set of test problems, provide a direction for evaluating parallel integration algorithms for nonlinear dynamics.

Besides the commonly used integration methods mentioned above, two predictor-corrector methods the Adams-Molton method and the Kunz method are implemented in parallel to solve the simple test problem. The evaluation of this group of methods will be performed in the future.

A specific plane stress problem is selected as another test problem. The problem has more realistic characteristic than the simple one-dimensional test problem. As the first step in performing experiments on this problem the parallel central difference and the parallel Newmark methods incorporating the parallel mixed Jacobi/Gauss-Seidel method are implemented and compared. The comparison provides verification for the conclusion mentioned in the first paragraph of this section. More integration methods and related evaluations are to be implemented on this test problem.

5.9 REFERENCES

5.1 L. Collatz, "The Numerical Treatment of Differential Equations", Spring-Verlag, New York, 1966.

5.2 N. Newmark, "A Method of Computation for Structural Dynamics", J. Engr. Mech. Div., ASCE, EM3, 67-94, 1959.

5.3 J. C. Houbolt, "A Recurrence-Matrix Solution for the Dynamic Response of Elastic Aircraft", NASA TN 2060, 1950.

5.4 E. L. Wilson, "A Computer Program for the Dynamic Stress Analysis of Underground Structures", Report No. SESM 68-1, Department of Civil Engineering, University of California, Berkeley, 1968.

5.5 D. Goehlich, L. Komzsik and R. Fulton, "Decomposition of Finite Element Matrics on Parallel Computers", ASME Computers in Engineering Conference, New York, NY, 1987.

5.6 R. Ou and R. Fulton, "Solution of Nonlinear Dynamic Response on Parallel Computers", AIAA Paper 88-2396, presented at 29th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Williamsburg, Virginia, 1988.

5.7 R. W. Hockney, "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis", J. Assoc. Comput. Mach. 12, 95-113, 1965.

5.8 R. W. Hockney and C. R. Jesshope, "Parallel Computers", Adam Hilger Ltd, Bristol, 1981.

5.9 C. F. Gerald and P. O. Wheatly, "Applied Numerical Analysis", Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

SECTION VI

CONCLUDING REMARKS

During the second year of study research has progressed along three paths:

1. Investigation of appropriate program designs and data structures for vector and parallel processing.
2. Investigation of data management requirements for MSC/NASTRAN to support parallel finite element computations.
3. Investigation of decoupled parallel numerical integration approaches for nonlinear dynamics.

The results of these research tasks are documented in this body of these reports. A summary is given below:

1. Two conceptually different program designs have been investigated to vectorize element dependent computations, i.e. single element vectorization and multiple element vectorization. Software testbeds have been established using the MSC/NASTRAN 10-noded TETRA element. Timing tests indicate superior performance of the multiple element method which is up to 5 times faster than the current MSC/NASTRAN code. Code redesign requirements and data structures have been documented with respect ot a future implementation in MSC/NASTRAN.
2. A feasibility study indicates that MSC/NASTRAN can evolve to a parallel system which may run effectively on computers with up to 16 processors. A design study for the linear static solution shows that major code redesign is necessary on the modul level to make MSC/.NASTRAN largely parallel. New parallel code designs, for EMG, EMA and SDR have been proposed. The file handling in a parallel environment has been investigated and a study for massively parallel FEM systems has been initiated.
3. Comparisons between the parallel explicit central difference method and parallel implicit methods have been made and initial evaluations performed. A comparison has also been made between a mixed Jacobi/Gauss-Seidel (MJGS) method and the parallel Cholesky decomposition method when

incorporated in integration procedures. The results show that MJGS has higher parallel effectiveness than the Cholesky. Parallel codes for solving a more realistic test problem - a plane stress problem - have been devised and some numerical experiments have been performed on this problem to compare the central difference method and the Newmark method. Besides the commonly used integration methods, two predictor-corrector methods have been implemented in parallel to explore new possibilities in parallel integration methods. New code designs implementing parallel integration approaches in MSC/NASTRAN have been studied.

4. Several papers were prepared reporting on the results of studies to date (ref. 6.1-6.7). An updated survey of new computer procedure trends shows a growing array of parallel/vector computers and the introduction of parallel/vector work stations.

It is recommended that followon work continue which builds on the work to date and which focuses on migrating MSC/NASTRAN into a well vectorized and largely parallel software system which will meet the computational challenges of the 1990's. It is also recommended that research be initiated which focuses on new code design for massively parallel systems.

In particular the following research tasks are recommended:

1. Vectorization of nonlinear force vector computations in MSC/NASTRAN.
2. Investigation of appropriate program designs and data structures for parallel nonlinear MSC/NASTRAN solutions.
3. Implementation of parallel numerical integration methods developed in this study into the DYNA-3D FEM code.
4. Investigate architectural aspects of a massively parallel FEM system. Study the potential of the DYNA-3D code as a basis for building a massively parallel special purpose FEM system.

These reserach tasks should lead to the following results:

1. Descripton of a vectorized NLEMG module including program design and data structures.
2. Outline of a program system architecture to perform efficient nonlinear finite element analysis on parallel computers.

3. Comparative assessment of the parallel performance of current and new integration algorithms for DYNA-3D for large order nonlinear finite element applications.
4. Assessment of the DYNA-3D code design with respect to its implementation in a massively parallel computing environment and by indicating necessary key software design changes.

6.1 REFERENCES

6.1 Goehlich, D., and Fulton, R. E. "Parallel Approaches to Nonlinear Finite Element Methods", ASCE Engineering Mechanics Division Conference, May 22-25, 1988, Blacksburg, VA

6.2 Goehlich D., Chiang, K. N. and Fulton, R. E., "Parallel Computer Approach to Finite Element Methods", UPCAEDM 1988 Conference, pp. 139-144, June 27-29, 1988, Atlanta, Georgia.

6.3 Ou, Rongfu and Fulton, R. E., "An Investigation of Parallel Numerical Methods for Nonlinear Dynamics", to be published in Journal of Computers and Structures, 1988.

6.4 Ou, Rongfu and Fulton, R. E., "Solution of Nonlinear Dynamics Response on Parallel Computers", AIAA Structures, Dynamics and Materials Conf., April 18-20, 1988, Williamsburg, VA.

6.5 Fulton, R. E., Chiang, K. N., and Goehlich, D., "Parallel Computer Implementation of Finite Element Methods", 2nd Internationa Conference on VEctor and Parallel Computing Issues in Applied Research and Development, Tromso, Norway, June 6-10, 1988.

6.6 Goehlich, D., Komzsik, L., and Fulton, R. E., "Parallel Linear Finite Element Method", to be published in the Journal of Computers and Structures, 1988.

6.7 Fulton, R. E., "Role of Supercomputers in Large Scale CAD/CAM Systems", 2nd ISR Supercomputing Conference on Visualization in Supercomputing, Tokyo, Japan, August 22-25, 1988.